Paper to be presented to the European Conference Product Data Technology, London,

A general framework for parametric product model within STEP and Parts Library[1]

Guy Pierra, Yamine Ait-Ameur, Frédéric Besnard, Patrick Girard, Jean-Claude Potier, LISI/ENSMA, Futuroscope, France

Keywords: STEP, P-LIB, parametrics, constraint-based model, variational geometry, CAD/CAM.

Abstract

This paper investigates the concept of parametrics in the perspective of exchanging parametric data models between heterogeneous systems either in the context of product data exchange (STEP), or in the context of parts library exchange (P-LIB). We first analyse the different approaches to parametrics and we propose a taxonomy based on their underlying mathematical structure. We then present the architecture of an information model that would enable the exchange of the different kinds of parametric models between different systems. We use the capabilities of the EXPRESS language and of the EXPRESS-G notations to formally define the proposed model. This model involves a four layers architecture and proposes to represent, and to exchange, not only the current instance and its parametric definition, but also an abstraction mechanism, called the dynamic context, that captures the invariant of all the possible instances, and the logical representation that provides an autonomous representation of these invariant. We then discuss the different kinds of parametric functions and we propose both a set of generic resources and a set of constraint-based parametric functions that might constitute a starting point for the standardization effort. This framework, that may also be used as a basis for system implementation, does not require any change in the existing (STEP) or emerging (P-LIB) Standards.

1. Introduction

STEP (Standard for the Exchange of Product Model Data, officially ISO standard 10303) is a series of international standards whose purpose is to define data across the full engineering and manufacturing life cycle. STEP is gaining wide acceptance in industry and its importance is becoming increasingly recognised.

Among the list of requirements stated in the early days were mentioned a product model core, application data requirements, data management, a mechanism for standard parts, *parametric* design features, and data syntax and file structure independent of the models content. Ten years later, the initial release of the STEP comprises twelve parts which, with reference to the original target, still omits parametrics.

In the mean time P-LIB (Parts Library, ISO 13584) has been developed [Pierra 94c]. This multi-part specification is a series of standards for the computer-sensitive representation and exchange of parts library data. Its objective is to provide a mechanism capable of transferring parts library data, independent of any application which is using a parts library data system. The nature of this description makes it suitable not only for the exchange of files containing parts, but also as a basis for implementing and sharing databases of parts library data.

The question of parametrics is crucial for such a topic, and a great deal of work has been undertaken in order to clarify the concept of parametrisation, and its relationship to the concept of a product. P-LIB adopts a generative perspective on parametrics. For P-LIB, parametrisation is related to the concept of a class. A product class is a set of products which are described together, which are assigned a common name, and whose instances may be distinguished from each other by the values of some parameters called product (or part) identification characteristics. Fixing the identification characteristics within the class fully identifies the product *instance*. A representation class is a set of representations which are described together and whose instances may be distinguished from each other by the values of some well defined parameters. Fix-

1.1 The research described in this paper was funded partially by EU under project ESPRIT III # 8984 (PLUS), and partially by the French Ministry of Industry under grant 93.4.930080.

ing these parameters within the class fully specifies the representation instance. A shape class, also called a (geometric) parametric model, is a representation class. Each instance shall be fully defined by its parameter values. To avoid data blow-up, the requirement of P-LIB is to capture globally the class level in such a way that every instance may be generated from this implicit description in a deterministic way.

Two years ago, a new Working Group was established within ISO TC 184/SC4 with the goal to provide parametrics capabilities to the different SC4 standards. The first step was difficult due to problems related to reaching an agreement about a common definition of the concept of parametrics, the requirements to be addressed, and the methodology to be followed. Progressively, the different kind of requirements became more precise, and it is now possible, to propose a global framework for introducing parametric capabilities both in STEP and in P-LIB.

The purpose of this paper is to propose such a framework. This framework is based on a previous proposal [Pierra 94a] [Pierra 94b]. It enlarges its scope to be able to integrate the requirements that emerged later on. This paper does not present a complete and finalised information model for parametrics: the technology is still not mature enough and continues to evolve. Its goal is more to clarify and to discuss the concept of parametrics, to provide hooks for later development, and to propose a complete approach for one of the clear requirements that emerged in particular from the P-LIB development team: the availability of a generative parametric data model.

This paper is organised as follows. In the next section, we propose a taxonomy of parametrics according to its underlying mathematical structure. This taxonomy enables a more precise identification of the requirements to be addressed. In the third section we discuss the commonalities of all the parametrics approaches. The very specific feature of parametrics is to gather the class level, in fact some kind of program either declarative or imperative, with one specific instance that represents one example of the program result. We call this example the *current instance*. We discuss the relationships between the program and the current instance, and we introduce the concept of dynamic context and its possible data model. We finally present a data model for parametrics variables and we propose a global architecture for parametrics modelling. In the fourth section of this paper we present a taxonomy of the required parametric functions. Canonical parametric functions enable to associate each attribute value with an expression that specifies this value. Whole-part modelling provides for a modular decomposition of parametric models. Constraint-based functions enable to create new items through their relationships with pre-existing items. Finally, control-structure functions provide for alternative or repetitive shape aspects in a class of shapes. For these different functions, data model architectures are defined. In the last section we briefly outline the implementation already achieved, and the on-going work.

In all this paper, we define formally the proposed information model using the EXPRESS language and the EXPRESS-G notations [ISO 10303-11]. We often reference the resources defined in Part 41, 42 and 43 of STEP [ISO 10303-41] [ISO 10303-42] [ISO 10303-43]. To characterize the entities which are formally defined by an EXPRESS information model (either in STEP or in our proposed parametric model), we write their names in **bold** font.

2. A mathematical taxonomy of parametric models

Over the last few years, a much work has been involved with ensuring more flexibility of a designed shape. The different approaches, often grouped under the name parametrics, address three very different problems.

2.1. Equality-based parametrics

The declarative approaches to parametrics [Sunde 87], and in particular variational geometry, consist of geometry problem solving: "given a model with a sufficient number of geometric constraints and a topological or approximate geometric description, we want the precise model to be evaluated automatically" [Roller 89];. The solution may be unknown to the user. He or she states the constraints. The role of the system is to compute the (possible) solution(s).

Mathematically, let P be the set of parameters defined over a domain *D*, P (often P = Rn), and let S (shape) be the set of variables for which the values (points, curves, numeric values,...) are required to describe an explicit instance (S belongs to the set S of all the possible shapes). A declarative model is an equation :

$$A(P, S) = 0; P \in D, S \in S ;$$

where A is an operator which is, generally, neither linear nor convex. We call such a structure an *equality-based parametric model.*

Many methods have been used to solve this problem: algebraic approaches, often based on the Newton-Raphson iterative method [Hillyard 78] [Light 82] [Lee 82], inference engine [Sunde 87] [Aldefeld 88], Buchberger's Gröbner Bases Method [Chou 84] [Chou 87], constraint graphs [Owen 91] [Bouma 95], and many others [Dufour 90] [Kin 89]. The popular 2D "sketcher", available on various CAD systems, corresponds to this approach.

Unfortunately, there are only partial solutions that address specific problems [Verroust 90]. Even if a lot of progress has been achieved since the precursory work of Sutherland [Sutherland 63], this approach has three intrinsic limits.

(1) The problem has, in general, exponentially many solutions. Even in the simple case of 2D points with distance constraints, each couple of constraints define two possible solutions. To over-constraint the model makes the problem NP-complete [Bouma 95]. It is therefore impossible to compute "the precise model". At the best "*one* precise model, or *all* precise models, may be computed.

(2) When the degree of some equations become greater than four it is impossible, in general, to compute all the solutions. Only *some* solutions may be (numerically) solved.

(3) To support only non-oriented constraints would forbid the use of all the procedurally-defined modelling constructs (e.g., sweeping, CSG, feature-based modelling) that leads to oriented constraints.

The example presented in figure 2.2, from [Bouma 95], shows that the first limit is a serious one. The same simple equality-based 2D parametric model may have very different kinds of solutions. Even if each specific sketcher uses specific heuristics, that often propose the "intended" solution, no general mechanism has yet been proposed that might ensure the determinism of the solving process, and, therefore, that the same solution is generated by two different solvers using the same set of equations.

These limits bound the present scope of equality-based parametric systems: to user-friendly design of 2D sketches, with often-successful or interactively-controlled solvers [Bouma 95], or to 3D solid model positioning [Kramer 92]. Progressively, some larger classes of problems will be covered, but, due to its underlying mathematical basis, this approach alone will never cover the whole set of requirements concerning parametrics.

2.2. Functional parametric model

The functional approach to parametrics, also referred to as constructive [Roller 89], addresses a very different problem: given a class of shapes whose design process is well known, and which may be supported by the interface of some design system, we want any instance, characterised by its parameter values, to be generated automatically in a deterministic way. We call such a structure a *functional parametric model*.

Mathematically, using the above notations, a parametric model is a function:

$$F: D\ S\ ;\ S = F(P)$$

Where F is the function that defines the instance from its parameter values.

Since the domain D is often non-specified, the function may "fail". This means that the parameter values do not belong to the domain D. Nevertheless, for all sets of parameter values that belong to D, the parametric model defines exactly one instance.

Various prototypes, compared in [Solano 94], and several commercial products are based on this approach. They address both 2D and 3D models. They may support feature-based design [Hoffman 92] [Roller 95], control structure with alternative or repetitive shapes, and they may generate variant programs [Pierra 94b].

An important characteristic of functional parametric models is that the function F is always expressed as a composition of functions:

$$F = fn\ fn\text{-}1\ ...\ f1$$

where each function duplicates, as part of its result, its input parameters:

$$fi : E1,i \times E2,i \times .. \times Ep,i \ \rightarrow\ E1,i \times E2,i \times .. \times Ep,i \times Ep+1,i.$$

We call the *fi* functions, the *parametric functions*.

The major limit of this approach is that the built-in parametric functions constitute the only (functional) constraint that may be captured. It is neither possible to constraint a circle to be tangent to three lines if such a (functional) constraint is not explicitly available, nor to specify cyclic constraints.

Moreover, for some 2D problems that may be easily specified through equality constraints, such as the one presented in figure 2.2, the specification as a functional parametric model is much less user-friendly (see fig. 2.6).

But this approach has one major interest. Provided that each parametric function is really specified as a function, i.e., with only one result (as it is done, for example in figure 2.3), the solving process is determinist and independent of the solver.

Besides its generative capabilities, that may be required in some contexts, this approach has two other interests:
(1) it may be used for any kind of modelling system, whether it is 2D or 3D, based on constructive solid geometry (CSG) or on Boundary representation (B-rep), and
(2) it may support very high level modelling constructs such as Boolean operation on B-rep models or feature-based modelling.

2.3. Inequality-based parametric model

A third category of parametric models involves only inequations. It may be stated, using the above notations, as:

$$A(P,S) \quad 0 \; ; P \in D, S \in S$$

Obviously, such a model is not intended to generate any instance. Capturing inequality constraints may address two requirements: (1) to control the allowed model modifications (for instance for design re-use purposes), or (2) to check whether the material product that results from a manufacturing process fits with a toleranced geometric model.

We quote such a model only because it shall be taken into account within a modelling architecture intended to support general parametrics capabilities.

2.4. Hybrid parametric model

The goal of the above taxonomy is to point out both the strengths and weaknesses of each mathematical approach to parametrics. As a result of the intrinsic weaknesses of each approach, few commercial products are restricted to only one of these approaches. Although all the 3D systems we know follow the functional approach for the 3D shape design constructs, most of them support equality-based parametric definitions of 2D-contours, and some of them support equality-based parametric positioning of 3D shapes.

From a data modelling point of view, this means than no parametric systems use *only* non oriented constraints. Therefore, the generic structure of a parametric data model may be defined as a set of constraints, each constraint requiring that some already existing entities are available (modelled in figure 2.1 through the **assumed** attribute of a **constraint**) and constraining a set of other entities (**defined** attribute). If different entities are involved in the **assumed** or **defined** attribute of some constraints, their roles are, in general, non identical. Therefore the **assumed** and the **defined** attributes shall correspond to a list (that may be empty for the **assumed** list).

A functional parametric data model is a subtype of such a generic structure. For this subtype:
1) each constraint is a function, and
2) the constraints are ordered (composition of functions), and
3) some variables, clearly identified, define the domain (**parameters**) of the global parametric function.
This is shown in the figure 2.1 using the EXPRESS-G symbolism defined in [ISO 10303-11]

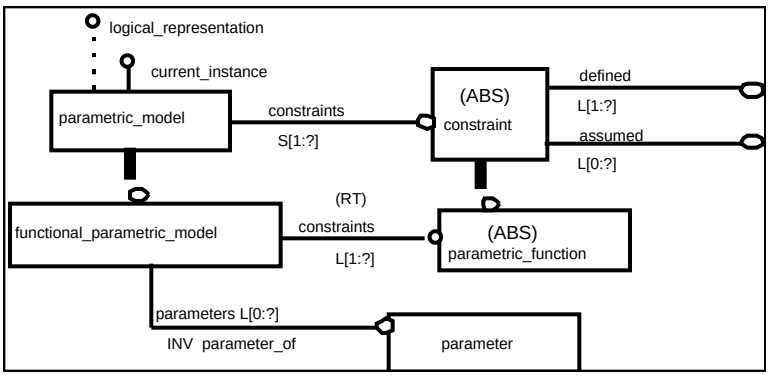


Fig. 2.1 : Planning model of a parametric information model

In this data model, both **constraint**s and **parametric_function**s are defined as abstract supertypes. They shall be specialised for each specific constraint or parametric function. The **parameter** entity is defined in

section 3.3 together with its relationship. The entities referred to by the **defined**, **assumed**, **current_instance** and **logical_representation** attributes are discussed below.

The goal of the framework defined in this paper is to support this kind of hybrid parametric data model. it also provides for possible specification of pure equality-based and/or inequality-based models.

2.5. Example

The following example, from [Bouma 95], illustrates the difference between equality-based and functional parametric models. The designer creates the sketch (a) and provides the following values: $d_1 = d_2 = 50$; $a_1 = a_2 = 30$; r = 30.

When the solver is invoked, it may return (according to its internal structure and heuristics) any of the three solutions (a) (b) or (c).

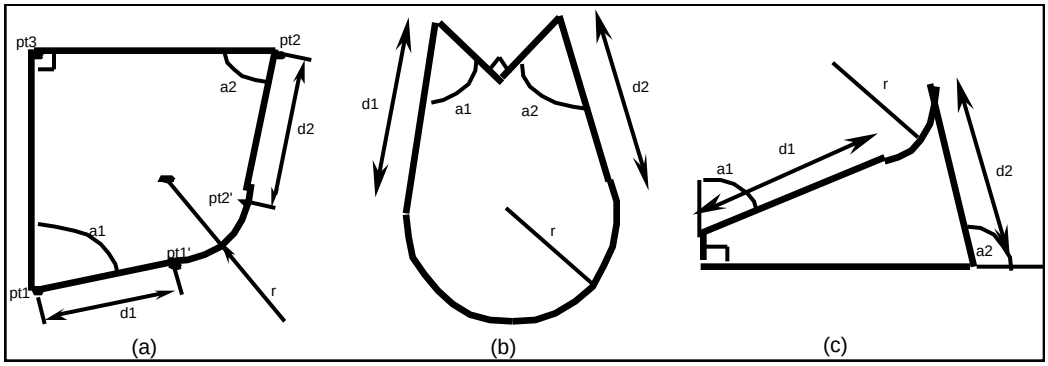


Fig. 2.2 : Three different solutions for the same equality-based constraint schema

Assuming, like [Bouma 95], that the user's intention is clearly to round the two adjacent segments, (irrespective of the angles a_1 and a_2), this parametric contour may also be specified, less easily but in an unambiguous way, as a functional parametric model.

We assume, here, that the functional parametric system supports the parametric functions specified in [ISO DIS 13584-31] (see figure 4.8), and that a display computer enables the user to specify graphically any kind of algebraic (or geometric-numeric, see section 4.1.3) expression.

It has already been underlined that parametric functions should be unambiguous, therefore each ambiguous geometric construct must be specified individually together with an ambiguity removal mechanism. For instance, in [ISO DIS 13584-31], ambiguity removal is based on entity orientation. Figure 2.3 graphically illustrates the specification given in that standard for the function "arc_fillet_2_ent" (that creates a fillet between two entities that may be trimmed lines or circular arcs) when the two entities are two trimmed lines.

The specification is the following:
(1) the two trimmed lines are to be shortened, and joined by a circular arc; this circular arc:
(2) is to be oriented from line 1 to line 2 (black arrows), and
(3) is to be the circular arc with the smaller radial angle that fulfils condition (2), and
(4) is to be the circular arc that shortens to the greatest extent the trimmed line, line 1 (first parameter of the function).

Such a specification, together with the capability (through a **same_sense** parameter in the parametric function domain) to specify for each involved entity whether the orientation to be considered is the actual entity orientation (**same_sense** = TRUE, grey arrow), or its opposite (**same_sense** = FALSE), enables an unambiguous definition of the parametric function "arc_filet_2_ent".

Obviously, this ambiguity removal process is hidden to the interactive system user: the system uses the picking ordering and positions (shown as 1 and 2 in figure 2.3) both to order the two involved trimmed lines and to generate the ambiguity removers.

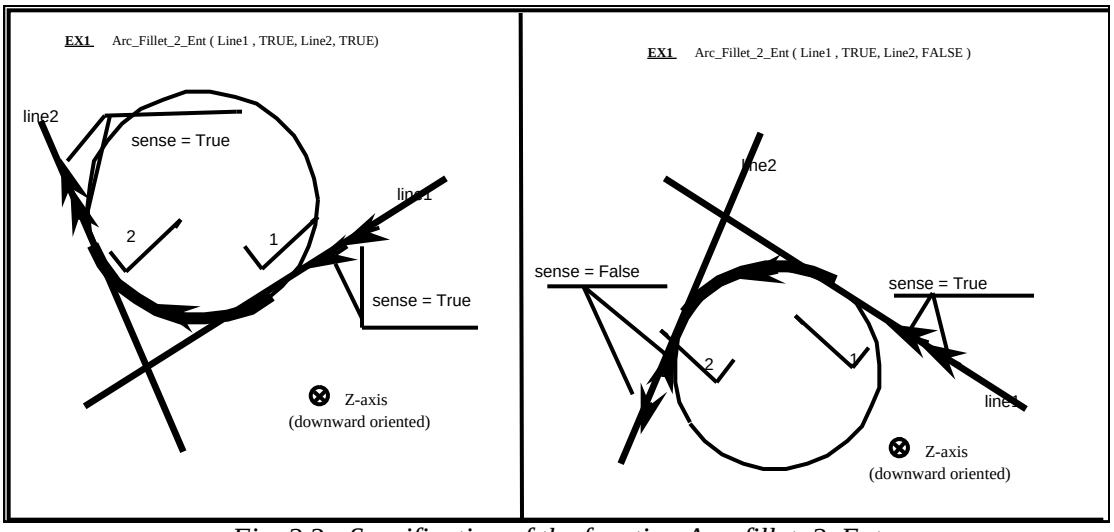


Fig. 2.3 : Specification of the function Arc_fillet_2_Ent

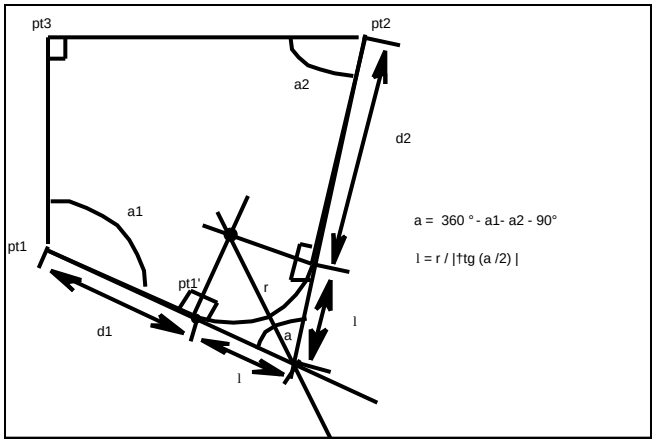
Using this function, the parametric contour shown in figure 2.2(a) may be specified interactively and unambiguously as a sequence of parametric functions.

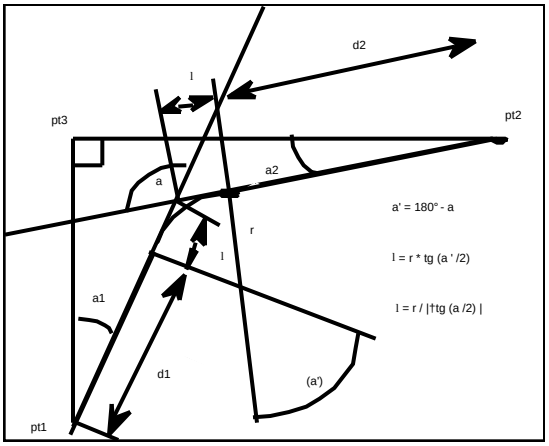
We first note that the circular-arc always shortens the two trimmed lines. Therefore, the designer draws, first, the two lines without the fillet (see figure 2.2(a)).

Figure 2.4(a) and 2.4(b) shows that the length of the two trimmed lines are to be extended by a length *l*. This length is defined in any case (see figure 2.4) by:

$$l = F(r; | tg(360 - a_1 - a_2 - 90) |)$$

(a)





(b)

Fig. 2.4: Hierarchical design of the contour

Using only the parametric functions specified in [ISO DIS 13584-31] (which deal with bounded curves: segment lines, circular arcs,...), the parametric contour may be built as shown in
figure 2.5 and may be recorded as the functional parametric model shown in figure 2.6.

As we will show below, this parametric model itself may then easily be captured as an EXPRESS data model using a meta-programming approach [Ait-Ameur 95] and, in particular, the expression schema defined in [ISO DIS 13584-20].

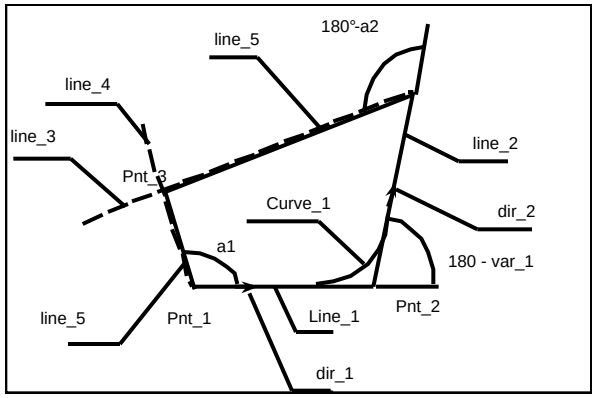


Fig 2.5 : The interactively designed contour

```
Pnt_1 := Create_Cartesien_Point (0, 0)
Var_1:= 360 - a1 - a2 - 90
Var_2 := r/ABS (TAN (Var_1 / 2))
Pnt _2:= Create_Cartesian_Point (d1 + Var_2, 0)
Line_1 := Create_Line_2_Pnt (Pnt_1, Pnt_2)
Dir _1 := Create_Direction_2_Pnt (Pnt_1, Pnt_2)
Dir _2:= Create_Direction_Angle (Pnt_2,  180  - Var_1)
Line_2 := Create_Line_Pnt_Length_Direction (Pnt_2 , d2 + Var_2, Dir_2)
Line_3 := Create_Line_Pnt_Length_Direction (Pnt_Extremity _Ent (Line_2), 180  - a2, 1000)
Line_4:= Create_Line_Pnt_Length_Direction (Pnt_1, a1, 1000)
Pnt_3 := Pnt_Intersection_2_Ent (Lin_3, Lin_4)
Line_5:= Create_Line_2_Pnt (Pnt_Extremity (Line_2), Pnt_3)
Line_6:= Create_Line_2_Pnt (Pnt_3, Pnt_1)
Curve_1:= Arc_Fillet_2_Ent (Line_1, TRUE, Line_2, TRUE)
```

Fig 2.6 : The functional parametric model

2.6 User interface issues

It should be noted that the way in which the constraints are captured from the user, and possibly displayed and modified, is orthogonal to the data structure taxonomy.

Even if functional parametric models are often captured during the design process (and therefore sometimes considered as history-based) and equality-based or inequality-based constraints often captured as a separated process, it is perfectly feasible for a parametric system:
1) to capture equality-based constraints during the design (some commercial systems already do this);
2) to edit functional constraints and to allow changes in a functional parametric definition;
3) to convert a functional parametric model into an equality-based parametric model, and (in general

through a dialogue with the user, resulting from ambiguity) to convert from equality-based to functional; and finally
4) to build in the mean time, two parametric models (e.g., one functional, the other equality-based) referring to the same edited representation (i.e., the same designed instance).

The user interface issues are therefore outside the scope of a data model for parametrics and they will not be considered in this paper.

3. Commonalities between the different parametric systems

Irrespective of be the underlying mathematical structure of a parametric system, all the systems share a common software architecture, and address several common issues. We first outline, in this section, the very specific feature of all the parametrics systems, namely the concept of current instance. We then discuss two major issues often referred to as topology changes and variable status. This allows us to introduce the proposed framework.

3.1. Two-level data models of parametrics systems

The principal characteristic of all parametric approaches is that the class level, which is in fact a program (either declarative, imperative, or constraint based), is always designed and represented with one instance. We call this instance the *current instance.* Figure 1 shows a very simple example based on numeric expression. The values which appear on the tree are the *current instance* values. In fact these values play two roles. (1) They explicitly represent the *current instance.* (2) These values stand for variables whose types may be deduced from the values.

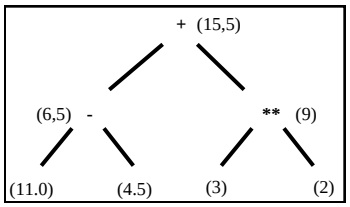


Fig. 3.1: A (parametric) expression interactively designed

Provided that the user builds this expression on the display computer of the CAD system, and provided that the expression itself is recorded, the (implicit) parametric program is perfectly defined (variable names are generated by the system):

real: x, y, z, result
integer: i, j, k
z := x - y;
k := ij;
result := z + k;

Fig. 3.2: The corresponding program

One difficulty remains. When the user reintroduces, in a subsequent expression, the value "15.5" on the display computer, the system can't decide whether it is a new "parameter", or it is the previously computed value.

Fortunately, this problem does not exist in geometry design. Since the geometric entities are stored in the CAD system database, the values for the *current instance* are database pointers. Picking up the same entity refers to the same database pointer (in the *current instance*), and therefore to the same variable in the (implicit) program.

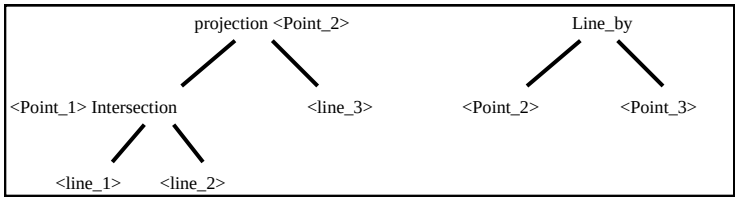


Fig. 3.3: A sequence of parametric constructs

When, in figure 3.3, <Point_2> is referred to within the second construct, the implicit variable is clearly identified.

Since numeric variables are needed in parametric geometric design, (at least for the explicit parameters of a functional parametric model) the same solution may be applied for the corresponding numeric values of the *current instance*. In parametric design, these numeric values are considered as database entities. They are stored in the database and they may be accessed via a menu or other dialogue conventions (e.g., dimensionning lines). Picking up such a value clearly identifies the (implicitly referenced) variable.

This dual structure of a parametric data model, on the one hand the current instance, on the other hand the constraints (functions, equations or inequations) that hold for any instance, and that are represented as relationships between the entities that constitute the current instance, suggests a straightforward data model for parametrics:
- the current instance is a (usual) STEP **representation** as defined in [ISO 10303-43],and
- this representation is parametric only because some more relationships are recorded between the representation items that constitute this representation.

If we decide that these relationships are modelled using the Entity-Relationship approach, and that no inverse attributes are required (which is true, at least at exchange time) we can introduce parametric capabilities within STEP without any change, neither of the STEP architecture, nor of the modelling tools (EXPRESS) or implementation forms already defined, nor of all the information models already standardised.

This approach that would consist, in figure 2.1, of assigning **representation_item** [ISO 10303-43] as the base type of the **defined** and **assumed** attributes, and **representation** as the type of the **current_instance** one, has already been proposed in [Pierra 94a] [Pierra 94b]. Examples of "STEP-compliant" data models based on this approach have been demonstrated (from now, as above, we will also use bold font to reference the items that are formally defined in some STEP EXPRESS model, quoting their source through a reference to an ISO 10303 document).

However, this data model, where each constraint directly refers to the values that constitute the current instance, can hardly be used as their internal data structure by parametric systems. A parametric system is intended to *modify* the current instance. If the relationship directly refers to current instance:
1) all the "current instances" shall have exactly the same structure, and
2) complex pointer manipulations are required when a new current instance is generated.

This problem has been widely studied in example-based programming [Halbert 84] [Myers 86] [Myers 90] [Girard 93]. The proposed mechanism, termed pointer variables or dynamic context [Girard 90] [Girard 93], consists in using indirect pointers.

Like standard programming environments, in example-based programming systems, the (implicit) pro-

grams contain variables, and the program (dynamic) context contains the values that are linked with the variables. The specific feature of these kind of systems is that both the dynamic context and the program are implicitly built when the user designs an example (i.e., the current instance). All created values (in the example) stand for implicit variable declarations (in the program context). All references to values (in the example) stand for reference to variables (in the implicit program). One main role of these systems is to manage the context of the program which ensures the indirect link between example values and program variables [Girard 93]. Figure 3.4 shows the dynamic context management of an example-based programming system.

A major issue when designing an information model for exchange of parametrics, is to decide whether or not the dynamic context is to be modelled, and, therefore, exchanged. We discuss in the next section the possible benefits of the exchange of such a mechanism.

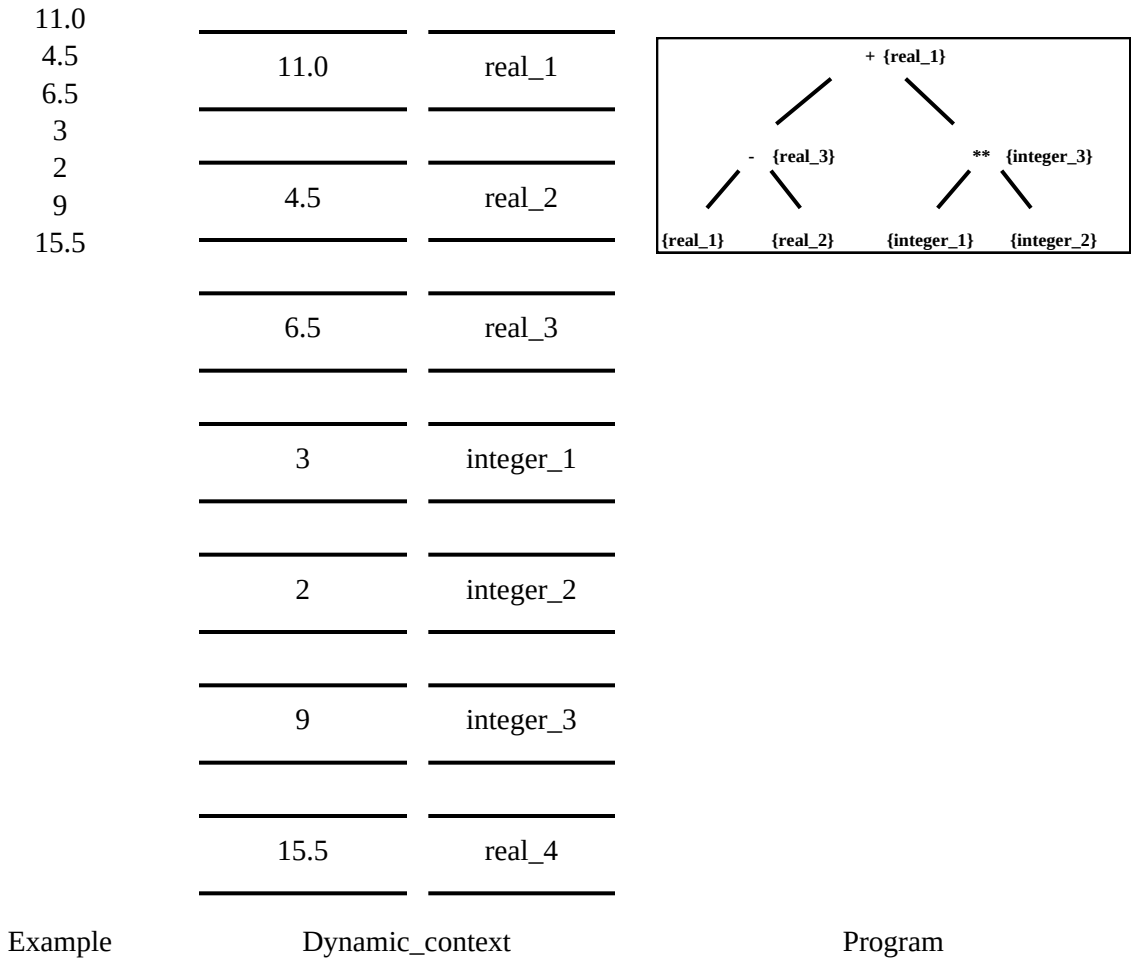


Fig. 3.4: Dynamic context management of example-based programming systems

3.2. Change in topology and concept of dynamic context

The mathematical models of parametrics discussed in section 1 highlight the major challenge of the parametrics CAD systems. A set of equations may be solved for different sets of values. A composition of functions may be computed for different input values within the domain of the global function. But the set of variables of the equations, or the domain and range of the different functions shall not change, else the problem is no longer defined. In terms of geometry, if the constraints directly refer to representation items, and if some representation items disappear or appear when a new current instance is generated, the parametric model is no longer defined. For instance, in feature-based modelling using a B-rep modeller, if a particular

constraint relates to a particular face, then if this face disappears, the constraint is no longer defined.

To support these kind of changes, we must identify the *invariant* of all the possible current instances, and allow constraints specification only between these invariants. Two solutions may first be considered, either to use geometric aggregates (a feature is a set of geometric representation items) or to use topology aggregates (a feature is represented as a set of open shells). But the capability to provide finer access (e.g., to the end face of a sweeped feature) needs a lower level of granularity, and a deep structuring of the invariants.

We propose to use in that intend a concept of dynamic context. A dynamic context consists of **parametric_reference**s (of which the information model is specified in figure 3.5). Each **parametric_reference** enables an invariant of a parametric model to be captured.

3.2.1. Underlying principles

The proposed information model is based on two general principles, with a third one specific to functional parametric models.

1 - Encapsulation principle

All the constraints shall refer only, either as their **assumed** or as their **defined** attribute, to **parametric_reference**s (or variables) that encapsulate the physical representation of the parametric model.

2 - Traceability principle

All the **representation_item**s that constitute the physical representation are referenced by one and only one **parametric_reference**.

The third principle addresses only functional parametric data models.

3 - Functional dependency principle

In a functional parametric model:
- each **parametric_reference** results from one and only one **parametric_function** or other **parametric_reference**
- all the **parametric_reference**s that belong to the **assumed** attribute of a **parametric_function** result from other **parametric_function**s that precede this parametric function in the **constraints** list of the corresponding **functional_parametric_model**.

3.2.2. Concept of parametric reference

A **parametric_reference** fulfils four roles:

1) It defines the lower level invariants, that result from its creation, and that may be accessed by other constraints. It is therefore a *structuring mechanism* of the parametric model invariants.

For instance, when a circular arc is created as a result of the arc_fillet_2_entities function, this function creates in the mean time (see [ISO 10303-42] for the definitions of the geometric representation items):
- the **circle** that is the **basis_curve** of this **trimmed_curve**;
- the **axis_2_placement** that is the position of this **circle**;
- the **point** that is the **location** of this **axis_2_placement**;
- possibly, the **point**(s) that are the **trim_1** and **trim_2** trimming point of the **trimmed_curve**;
- possibly, the **direction**s that are the **axis** and **ref_direction** of the position **axis_2_placement**.

Taking into account that the constraints expressed as parametric functions generally result in the creation of a highly structured set of **representation_item**s (e.g., a feature in a B-rep model), a **parametric_reference** enables a structured set of referencable items to be created.

2) A **parametric_reference** records the virtual entities that are involved in a parametric model (e.g., a feature) independently of the physical **representation_item**s that represent these entities in the current instance. It is therefore an *abstraction mechanism* for the parametric model invariants.

For instance, a 2D parametric model may contain an optional fillet. This fillet shall exist only if a particular length-measure is greater than a particular value. A **parametric_reference** that stands for this optional fillet may be involved in, e.g., a symmetry, even if the fillet does not exist in the current instance.

Another example is provided by feature-based modelling. When a depression feature is involved in a design, the model invariant is the virtual feature itself, and not the set of faces, edges and vertices that represent the regularised Boolean difference of a solid model with the virtual feature.

To achieve this role a **parametric_reference** may refer, through its **logical** attribute, to **representation_item**s that do not belong to the current-instance **representation**, but to another **representation**, called the *logical representation*.

3) A **parametric_reference** controls the level of granularity of the items that may be accessed by other constraints. It is therefore a *control mechanism of the referencable items.* To achieve this role:
- each subtype of **parametric_reference** defines, through its **referencable_item**s (derived) attribute, the other **parametric_reference**s or variables that logically constitute it, and that may be accessed from the latter constraints;
- all the items in the **referencable_item**s attribute are declared as OPTIONAL; if some item is not present, this means that it is not accessible as a logical constituent of the **parametric_reference** (either it exists independently, or it is hidden because it is not considered as an invariant).

For instance, a **circular_arc_ref** provides an OPTIONAL access to the **circle** that constitutes its **basic_curve**, and to its **trim_1** and **trim_2 point**s. If the **circle** is not provided, this means that this circle is not part of this **parametric_reference**. The same **circular_arc_ref** entity may therefore be used both to define the **defined** attribute of the parametric function arc_fillet_2_ent (that creates both a new **circle** and a new **axis_2_placement**), and the **defined** attribute of the parametric function **arc_circle_2_angle** (that creates only a circular arc using an existing **circle**),

4) A **parametric_reference** refers, through its **physical_item**s attribute or by means of the **physical_items** attribute of the other **parametric_reference**s that belong to its **referenceable_items** attribute, to all the **representation_item**s that represent it in the current instance. It is therefore a *traceability mechanism* (and an interactive-user-access mechanism) for all the **representation_item**s of the current instance.

3.2.3 Data model of a parametric reference

The following EXPRESS specification defines the information model of a **parametric_reference** (the **generic_variable** entity is discussed in section 3.3; the **constraint** entity is defined, using EXPRESS-G [ISO 10303-11] in figure 2.1).

```
TYPE param_ref_or_var_select = SELECT (parametric_reference, generic_variable);
END_TYPE;

TYPE param_ref_or_constraints_select= SELECT (parametric_reference, constraint);
END_TYPE;

ENTITY parametric_reference
ABSTRACT SUPERTYPE;
```

```
   recorded_logical_items:OPTIONAL LIST[1:?] OF UNIQUE representation_item;
   physical_items        :LIST[0:?]OF UNIQUE representation_item;
   referencable_items    :LIST[0:?] OF UNIQUE param_ref_or_var_select;
   definition_reference  :SET OF param_ref_or_constraints_select;
DERIVE
   logical_items         :LIST[1:?] OF representation_item
                         :=NVL (recorded_logical_items, physical_items);
END_ENTITY;
```

Fig. 3.5 Information model of a parametric model invariant

The **physical_items** and **logical_items** capture the actual and virtual representation of the invariant. When both are identical, only the actual representation is recorded. In this case, the **logical_items** attribute equals the **physical_items** one (it is the meaning of the NulL Value -NVL- function: if the first argument has no value, then the value of the second one is returned). Both the **physical_items** and **logical_items attributes are** list-ordered because each subtype of **parametric_reference** shall contain, as it **logical_items [1]** attribute, the **representation_item** of which the **parametric_reference** is an abstraction.

The **referencable_items** attribute captures the whole set of **parametric_reference**s and **generic_variable**s (from [ISO DIS 13584-20]) that result from its definition, and that may be accessed by other constraints. In a functional parametric model, this attribute defines the new **parametric_reference**s or **generic_variable**s that are allowed for use in the latter **parametric_function**s. This attribute is intended to be derived in any subtype of **parametric_reference**.

The **definition_reference** attribute captures the whole set of constraints and **parametric_reference**s that contribute to the parametric definition of the item encapsulated by a **parametric_reference**. In a functional parametric model, a global rule ensures that the size of this set equals 1 and that all the **parametric_reference**s that belong to the **assumed** attribute of a **parametric_function** result from previous parametric functions according to the list-order defined by the functional parametric model. This attribute is also intended to be derived in any subtype of **parametric_reference** and computed from the inverse relationships.

3.2.4 parametric reference for geometry models

Each time a new category of **parametric_reference** is defined, derivation functions for both **referencable_items** and **definition_reference** shall be specified. According to the rules of the EXPRESS language, such derivation functions may not be redefined in the inheritance tree. Therefore, we define a subtype that constitutes the root of the whole sub-tree of **parametric_reference**s corresponding to parametric geometry. This entity is associated with two global functions that compute these attributes for the whole supertype/subtype sub-tree of **parametric_reference**s corresponding to the various [ISO 10303-42] **representation_item**s.

```
ENTITY geometry_parametric_reference
ABSTRACT SUPERTYPE
SUBTYPE OF (parametric_reference);
DERIVE
   referencable_items    :LIST[0:?] OF parametric_reference
                         := compute_geom__ref_items(SELF);
   definition_reference  :SET OF parametric_ref_or_constraints_select
                         :=compute_geom__def_reference(SELF);
END_ENTITY;
```

Fig. 3.6 Root of the parametric reference entities for parametric geometry

The **parametric_reference** entity shall be sub-typed for each type of **representation_item** intended to be referenced by a constraint. This means, in particular for parametric geometry, that

geometry_parametric_reference shall be subtyped for each type of **geometric_representation_item** and **topology_representation_item** specified in [ISO 10303-42], according to an identical subtype/supertype network, but with a specialisation of the two remaining explicit attributes of **parametric_reference**.

The following rules define the systematic process used to associate with each **representation_item** its corresponding **parametric_reference**:

•1) For each **representation_item** intended to be parametrically defined, a corresponding subtype of **parametric_reference** is defined.

•2) Each **parametric_reference** subtype takes the same name as the corresponding **representation_item**, post-fixed by "_ref".

•3) Each attribute, whose type is associated with a **representation_item** entity data type or aggregate of entity data types, is represented by an attribute of the same name whose optional value is defined as the **parametric_reference**s subtypes that correspond to this entity data type

•4) Each attribute whose value is a number, a Boolean or a string is not represented (it may be queried from the **logical_items[1]** attribute)

•5) Each attribute whose type is a select type that involves both a simple type and a **representation_item** entity data type is represented as an attribute of the same name of which the data type is an (OPTIONAL) **parametric_ref**.

Other particular subtypes are defined:
- for the specialisation whose types are required to specify the domain of some parametric functions (e.g., **trimmed_line_ref** and **circular_arc_ref** for 2D parametric model)
- for the logical entities that appear only in parametric geometry (e.g., **parametric_feature_ref**, **pattern_ref** for simple repetitive shapes, for optional shapes, ...).

Each **parametric_reference** subtype declares the referencable items it may consist of. These referencable items are collected in the derived **referencable_item**s attribute by the global function **compute_geometry_ref_items**.

For instance, the parametric reference that represents a referencable **trimmed_curve** (figure 3.7), that specialises **parametric_reference** through the path (defined from [ISO 10303-42] and [ISO 10303-43]): **bounded_curve_ref**, **curve_ref**, **geometric_representation_item_ref**, **geometry_parametric_reference**, **representation_item_ref**, **parametric_reference**:
- declares as (optional) **referencable_items** its basis curve and trimming points (they are collected in the derived **referencable_items** attribute by the global derivation function),
- specialises its **recorded_logical_items** as a trimmed curve,
- specialises its **physical_item**s attribute as a list of trimmed curves that share the same basis curve (the trimmed curve of the current instance may be modified by modifying feature [Hoffman 92] such as a fillet)

```
     ENTITY trimmed_curve_ref
     SUBTYPE OF (bounded_curve_ref);
        SELF\parametric_reference.recorded_logical_items
                              :OPTIONAL LIST[1:1] OF trimmed_curve;
        SELF\parametric_reference.physical_items
                              :LIST[0:?] OF trimmed_curve;
           basis_curve        :OPTIONAL curve_ref;
           trim_1             :OPTIONAL point_ref;
           trim_2             :OPTIONAL point_ref;
        WHERE
           WR1:(*same basis_curve for the different physical items*)
END_ENTITY;
```

Fig. 3.7 Example of parametric reference

Such a parametric reference is further specialised into a **circular_arc_ref** that specialises the **basis_curve** into a **circle_ref** and constrains the referenced **trimmed_curve**s to have **circle**s as their **basis_curve** (see [ISO 10303-42]).

New categories of parametric references may be added by defining new subtypes of the generic resource **parametric_reference** and by defining new specific derivation functions that compute the **referencable_item** and **definition_reference** attributes across the whole structure of this new parametric references sub-tree. This mechanism provides for subsequent standardisation of new parametric capabilities (e.g., parametric form feature) when the technology is mature enough to enable a consensus to be reached.

3.3 Concept of a variable

Besides the reference to **representation_item**s, a parametric model also uses variables.

In computer science, a variable consists of three parts:
- 1) a syntactical representation that provides a name that enables the variable to be referenced and that specifies its type of allowed values,
- 2) a mechanism, usually termed a context (in imperative programming) or an environment (in functional programming) that generates by some means (it may be, e.g., stored) the value of this syntactical representation,
- 3) a function, usually called interpretation function, which bounds the value mechanism to the syntactical representation.

This threefold concept is modelled in [ISO DIS 13584-20] by a threefold data model presented in Figure 3.8.
- 1) A **generic_variable** entity captures the syntactical representation of a variable and defines, by subtyping, its allowed type of value. This entity is referenced by e.g., expression, functions,...
- 2) A **variable_semantics** entity captures the mechanism that represents a value.
- 3) A relationship, termed **environment**, associates a **variable_semantics** with a **generic_variable**.

The **variable_semantic**s data type is an abstract data type that shall be subtyped each time a new mechanism is to be captured, e. g., the "SELF" keyword of object-oriented languages (see [ISO CD 13584-24]), the X-coordinate in some well-defined Cartesian coordinate reference system).

In parametric data models, we want to capture two semantics:
- 1) the concept of the **internal_variable**; the value of such a variable results from the constraints (or parametric functions) that contain its syntactical representation in their **defined** attribute, and
- 2) the concept of the formal **parameter** of a (functional) parametric model; such a variable should not belong to the **defined** attribute of any (internal) constraint; its value results from an external mechanism that assigns an actual value to a formal parameter when and where the (functional) parametric model is involved.

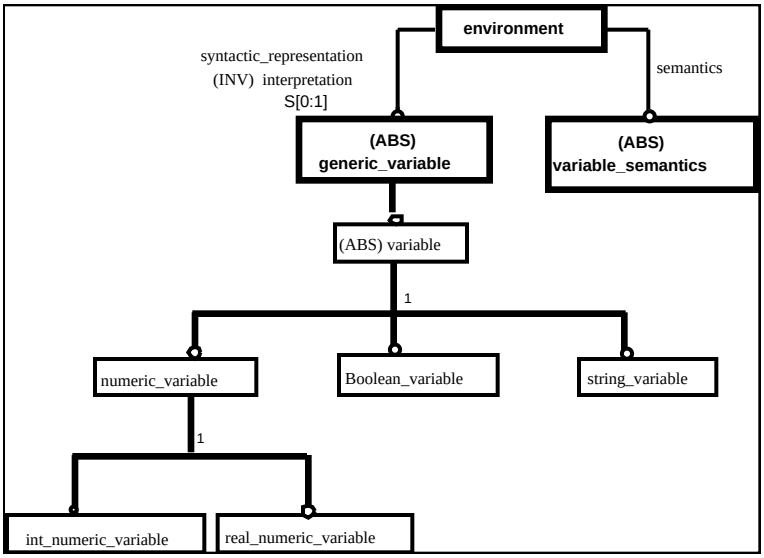


Fig. 3.8 Simplified data model of variable (from [ISO DIS 13584-20])

From a structural point of view, both types of **variable_semantics**:
- are associated with a name (possibly automatically generated);
- may be associated with a text, that describes its role,
- may be associated with a linear or angle dimensioning that enables to graphically represent its meaning, and
- is possibly associated with a value that represents the value for the current instance.

Figure 3.9 presents a planning model of the variables involved in parametric modelling.

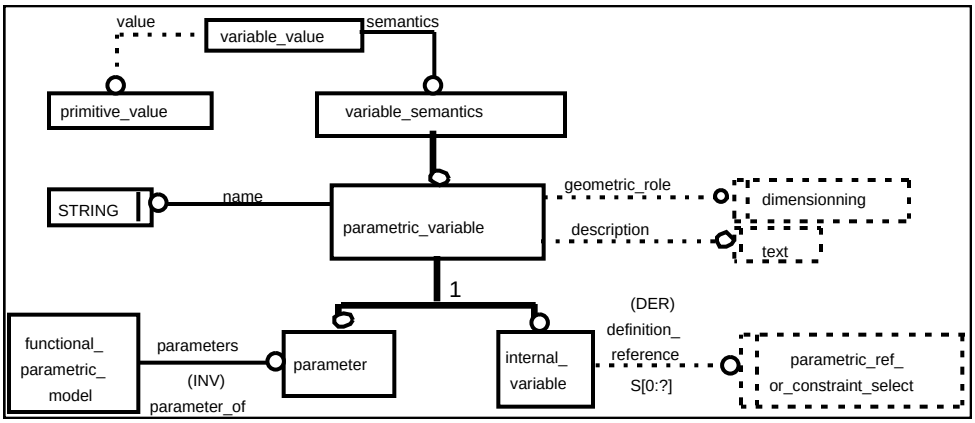


Fig. 3.9 Planning model of the variables in parametric modelling
(for clarity, the referenced entities are incompletely defined)

In this planning model, **text** is referenced from the **support_resource_schema** [ISO 10303-41] and **primitive_value** from the **instance_schema** [ISO CD 13584-24]. The complete model of **parametric_model** is presented in the next section.

3.4 Four layers architecture of a parametric model

The overall architecture of the proposed framework is defined in figure 3.10. A **parametric_model** is a **representation** (from [ISO 10303-43]) of which the **item**s and **representation_context** inherited attributes are

derived and equal to the ones of the current instance. A **functional_parametric_model** is a **parametric_model** where some rules ensure that the functional dependency principle (see section 3.2.1) is fulfilled.

For clarity, the logical representation and the current instance are represented separately as two different ISO 10303-43-**representation**s, and the **environment** and **variable_value** relationships and most of the derived attributes are not represented.

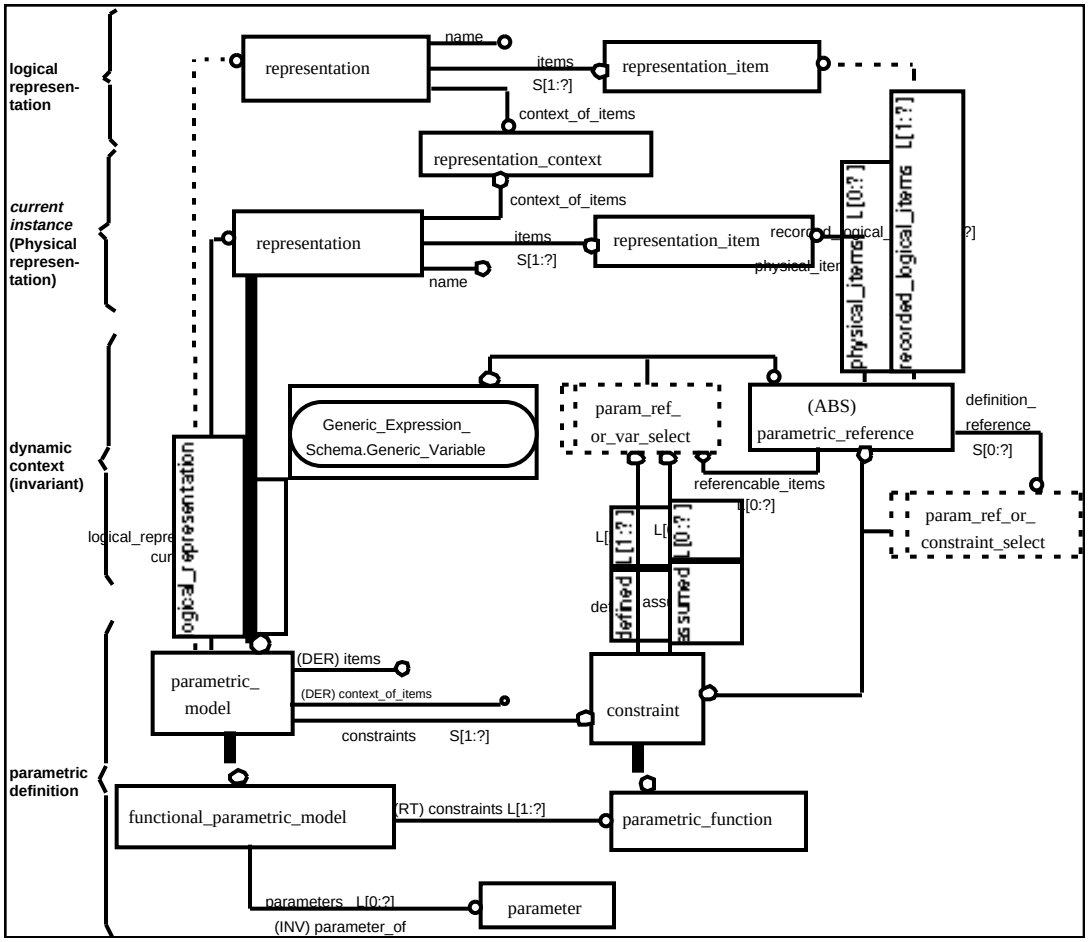


Fig. 3.10 Four-layer architecture of the proposed framework (Planning model)

This diagram emphasises the four layers architecture of the proposed framework:
- The current instance layer is a usual STEP **representation**. It is only referenced by one (or several) parametric definitions by means of a dynamic context.
- The dynamic context layer represents the structural invariant of all the possible instances of the parametric model.
- The parametric definition layer provides a parametric definition (of any kind, and possibly of different kinds) of the same current instance.
- The logical representation layer may or may not exist. Its role is to capture the logical items involved in a parametric definition when these logical items are not completely represented in the current instance, or when they have been modified after their creation by a constraint. When it exists, the logical representation shall use the same **representation_context** as the current instance.

The required parametric functions are discussed in the next section.

4. Taxonomy of needed parametric functions

We already pointed out that parametric functions are special cases (in the data model: subtypes) of constraints, and may therefore be used in any kind of parametric system. Moreover, the main requirements from Parts library is the availability of functional parametric model. We have therefore concentrated our work on this kind of constraints. We present in this section both a taxonomy of the required parametric function and:
1) some generic resources for expressions and whole-part modelling, and
2) a set of constraint-based parametric functions that address our specific requirement.

4.1 Expressions

Expression is one of the generic resources required for all kinds of parametric models. Expressions are directed acyclic graphs whose internal nodes are operators, and whose leaves, in parametric modelling, may be literal values, variables, or entities of the geometric model. Our parametric data model USEs all the resource constructs defined in [ISO DIS 13584-20] (**generic_expressions_schema** and **expressions_schema**).

4.1.1 The different types of expression for parametrics

Our parametric data model provides for four types of expressions. **numeric_expression**s, **Boolean_expression**s and **string_expression**s are the resource constructs defined in [ISO DIS 13584-20]. They enable the use of all the operators defined in the EXPRESS reference manual [ISO 10303-11] for simple types. The fourth subtype of **generic_expression**, called **parametric_ref_expression** is specific to parametrics. It models the **generic_expression**s that return a **parametric_reference** entity.

```
ENTITY parametric_ref_expression
   SUBTYPE OF (generic_expression);
END_ENTITY;
```

A **parametric_ref_litteral** is a special case of **parametric_ref_expression** that represents an existing **parametric_reference**.

```
ENTITY parametric_ref_litteral
   SUBTYPE OF (parametric_ref_expression, generic_litteral);
   the_value: parametric_reference;
END_ENTITY;
```

4.1.2 Instance access operator

The instance access operator is a 3-ary operator that provides a logical access to the attributes of the current instance while controlling that this access involves only invariant, i.e., **parametric_reference**s. It is defined as a subtype of the **m_ary_gen_expression** of the generic expression schema and it takes three operands:
- a **parametric_ref_expression**, that evaluates to a **parametric_reference** that encapsulates the accessed entity,
- a **string_expression**, that specifies the name of the accessed attribute in the format of an uppercase string, and
- an optional list of **numeric_expression**s, that specifies, when the attribute is an aggregate (or an aggregate of aggregates, ...), the referenced items. All these optional **numeric_expression**s shall evaluate to integer.

The instance access operator returns a value of any one of the types: **parametric_reference**, real, integer, string or Boolean.

```
ENTITY instance_access_operator
ABSTRACT SUPERTYPE OF (int_instance_access_operator,
                 real_instance_access_operator,
                 string_instance_access_operator,
                 boolean_instance_access_operator
                 parametric_ref_instance_access_operator)
SUBTYPE OF (m_ary_gen_expression);
   access_to: parametric_ref_expression;
   attribute: string_expression;
   index: OPTIONAL LIST [1:?] OF numeric_expression;
END_ENTITY;
```

Five subtypes enable to specify the type of result of an **instance_access_operator**. These subtypes provide for strong type checking and allow to involve the result of an **instance_access_operator** in a simple **expression** conforming to the **expression_schema** [ISO DIS 13584-20]. They are declared for instance as:

```
ENTITY int_instance_access_operator
   SUBTYPE OF (integer_defined_function, instance_access_operator);
END_ENTITY;
```

and

```
ENTITY parametric_ref_instance_access_operator
   SUBTYPE OF (parametric_ref_expression, instance_access_operator);
END_ENTITY;
```

The role of these operators is twofold:
•1) to *compute* the value of any simple attribute or member of an aggregate attribute of a **parametric_reference** that corresponds either to a **parametric_reference** or to a simple value, and,
•2) to capture a reference by its role, and not by its value.

To achieve this goal this operator processes as follows:
- if the referenced attribute refers to a **parametric_reference** return it;
- if the referenced attribute is a simple type value, return it;
- if the referenced attribute is not defined in the **parametric_reference** data type, or is defined but has no value, access to the first attribute of the **logical_items** list and query the same attribute name, then:
• if this attribute does not exist, return UNKNOWN,
• if the attribute exists and is of simple data type, return it,
• if the attribute exists and is of **representation_item** data type, return the **parametric_reference** that encapsulates this entity by referencing it in its **logical_item**s list.

This operator enables for instance:

(1) When a **circular_arc** is created as a fillet between two entities, to access to its radius through the expression (defined, in the exchange file conforming to [ISO 10303-21], as a tree - see [ISO DIS 13584-20]):
real_instance_access_operator
(
parametric_ref_instance_access_operator(#<integer>, 'RADIUS', $),
'ITS_CIRCLE', $
)

(2) When a circle is created by circle_radius_a2p [ISO 13584-31] (defined by its position and its radius) to access to its centre *as the centre of this circle*, and not as the point that represents it in the current instance (capture the user intend independently of the structure of the current instance). If the circle is latter redefined through a constraint-based definition, the reference will represents the centre of the new circle.

4.1.3 Geometric-numeric operators

Geometric_numeric_operators are operators whose domains are defined as a set of **geometric_representation_item**s or **topological_representation_item**s [ISO 10303-42] and that compute a real value.

Seven such operators (declared as subtypes of **numeric_defined_function** USEd from [ISO DIS 13584-20]) are defined. Their signatures are:

distance : point_ref_or_vertex_ref x point_ref_or_vertex_ref ->REAL
x_coord : point_ref_or_vertex_ref->REAL
y_coord : point_ref_or_vertex_ref->REAL
z_coord : point_ref_or_vertex_ref->REAL
angle : linear_ent_ref x linear_ent_ref-> REAL
start_angle : circular_arc_ref -> REAL
end_angle : circular_arc_ref -> REAL

Where **linear_ent_ref** is a select type that stands for **direction_ref**, **line_ref**, **plan_ref**, **trimmed_curve** of which the **basis_curve** is a line or one of the different **bounded_surface**s of which the basis surface is a plane.

4.1.4 Generic expressions

As a result of these operators, an instance of a **generic_expression** in our parametric model is a recursive structure that may contain any number of **instance_access_operator**, **geometric_numeric_operator**s and operators defined for number type, string type and Boolean type in the EXPRESS reference manual [ISO 10303-11].

Two functions are associated with a **generic_expression**:
- the **used_gen_variable**s function (specified in [ISO DIS 10303-20]) computes all the **variables** used in a **generic_expression** tree, and
- the **pm_used_item** function (defined in our parametric model) computes all the **parametric_reference**s used in a **generic_expression** tree.

These functions enable to assert that, in a functional parametric model, no forward references are done.

The correctness of a **generic_expression** is checked at run-time. If an expression is not correct (possibly as a result of the values assigned to the parameters of the current instance), this only means that the functional parametric model fails. The user dialogue to be possibly implemented in this case is outside the scope of this data model.

4.2 Canonical parametric function

The first subtype of **parametric_function** required for any kind of parametric modelling is the function that associates with each attribute of a parametrically-defined **representation_item**, the specification of its value through a **generic_expression**, or, when the value of some attribute is an aggregate, trough an aggregate of **generic_expression**s. We call such a parametric function a *canonical parametric function.*

Different data models may be considered to capture such a parametric function. One method, proposed in [PIERRA 94a], consists in defining a tree structure of canonical parametric functions. Each canonical parametric function corresponds one to one to each **representation_item** data type. Its definition and subtype/supertype structure is systematically derived from the definition and subtype/supertype structure of the corresponding **representation_item** data type (see [PIERRA 94a]).

Another method, that we propose in this paper (see figure 4.1), consists in defining one unique canonical parametric function that applies to any parametrically-defined representation item, with the specification of a mechanism to relate each **generic_expression** with a well-identified attribute of the parametrically-defined **representation_item**. The powerful expression structure defined in section 4.1 provides a large expressive power to this canonical parametric function.

The basic idea of this approach is to use pattern matching with the physical file structure (conforming to [ISO 10303-21]) of the created **representation_item**s. We note that a canonical parametric function only defines one **representation_item**, that this **representation_item** is encapsulated by one **parametric_reference** that refers to it through its **logical_items[1]** attribute, and that this **parametric_reference** does not contain any other **parametric_reference**. Therefore, the structure of the corresponding **representation_item** in the current instance (or logical instance) may be used for matching each **generic_expression** with the attribute value it corresponds to**.** The data model is as follows:

```
ENTITY canonical_parametric_function
SUBTYPE OF parametric_function;
   SELF\constraint.defined: LIST[1:1]OF parametric_reference;
   Instance_spec: simple_or_complex_instance_spec;
DERIVED
   used_var        : SET OF generic_variables
                   :=used_gen_variables_in_instance_spec(SELF.instance_spec);
   used_ref        : SET OF parametric_reference
                   :=used_param_ref_in_instance_spec(SELF.instance_spec);
   SELF\constraint.assumed:LIST[0:?] OF param_ref_or_var
                   :=set_to_list(SELF.used_ref + SELF.used_var);
WHERE
   WR1: SELF\parametric_function.defined[1].referencable_items=[];
END_ENTITY;
```

Figure 4.1 Proposed data model for canonical parametric functions

The set of **generic_expression**s is structured according to the **simple_or_complex_instance_spec** data type that enables to associates unambiguously each **generic_expression** with each attribute of a **representation_item**. The following EXPRESS-G diagram defines the data model of a **simple_or_complex_instance_spec**.

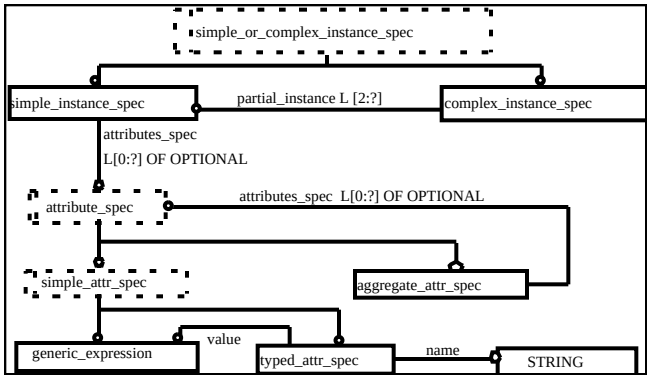


Fig. 4.2 Structured set of generic expressions associated with the attributes of a parametric reference instance

The association of the **generic_expression**s is done as follows.

1) Compute the evaluated set that correspond to the subtype/supertype expression in which is involved the unique **SELF\parametric_function.defined[1].logical_items[1].representa-tion_item** instance, as specified in Annex B of [ISO 10303-11].

2) Determine the mapping rules to apply to this instance, as specified in [ISO 10303-21], clause 11.2.5.1. If these rules define an internal mapping, the **simple_or_complex_instance_spec** shall be a **simple_instance_spec**, else, it shall be a **complex_instance_spec**.

Then associate each attribute that should explicitly appear in a physical file conforming to [ISO 10303-21], conformance class 1, to describe this instance with a **generic_expression** of the **instance_spec** attribute following the tree structure presented for **simple_or_complex_instance_spec** in figure 4.2.

Each attribute that should appear as an undetermined ('?') value shall be associated with an undeterrmined expression. Each attribute that should appear as an integer shall be associated with a **numeric_expression** that evaluates to integer value. Each attribute that should appear as real, Boolean, string or entity name shall be associated with a **numeric_expression**, **Boolean_expression**, **string_expression** or **parametric_ref_expression** respectively. Each attribute of aggregate data type shall be associated with a list of **generic_expression**s of which the base type matches with the base type of the aggregate, as defined above for simple values. If the attribute data type is either a list or an array, each generic expression correspond one to one to each value of the aggregate. If the attribute data type is either a bag, or a set, the number of **generic_expression**s shall be equal to the number of members of the aggregate, but the list order of the **generic_expression**s is meaningless.

4.3 Assembly design and whole/part modelling

The second kind of parametric functions, that we call parametric whole/part modelling, are intended to provide the capability to insert a parametric model (or a non parametric **representation**) within an embedding parametric model (or non parametric **representation**). We call the embedded parametric model an occurrence and the embedding parametric model a (instance of) parametric model. Parametric whole/part modelling defines a recursive representation structure where instances are inserted as occurrences within the context of other instances.

Parametric whole/part modelling involves a three level architecture where the parametric model, that constitutes the class level represents a family of cognate representations (e.g., all the possible blocks), where the instance level represents one particular representation of this family (e.g., one block of a given size) and where the occurrence level represents its insertion within an embedding representation (e.g., as a **mapped_item** from [ISO 10303-43]).

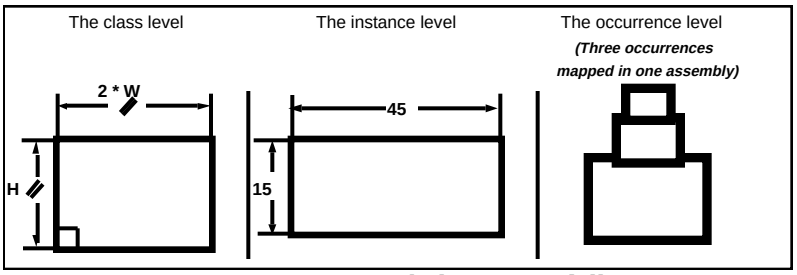


Fig 4.3 Parametric whole-part modelling

The very specific feature of parametrics, namely the fact that the class level is, in the mean time, an instance through the concept of current instance, enables to consider two different possible data models for these three levels.

In any cases, parametric whole-part modelling shall ensure the following:

•1) An occurrence shall be re-evaluated each time the embedding instance is re-evaluated to ensure that the

constraints that specify both the positioning (location and orientation) and the dimensioning of the occurrence by reference to the embedding instance still hold.
•2) All the references from the instance to some **representation_item**s of the occurrence shall be preserved despite this re-evaluation.

The structure of a parametric whole/part model is therefore heavily dependant on:

•1) the constraints that specify the occurrence by reference to its embedding instance, and
•2) the level of granularity of the occurrence items that are accessible, for reference, from its embedding context.

The major challenge of parametric whole/part modelling, that is still a very competitive process, is to achieve four objectives:
•1) to enable a great flexibility for occurrence specification,
•2) to provide a fine grain access (e.g., every **representation_item**) to occurrence internal representation,
•3) to ensure a re-evaluation process as deterministic as possible, and, finally
•4) to ensure a fast re-evaluation of the complete model

These objectives are conflicting with each other: the more flexible solution that would be to consider the complete whole/part structure as a global equality-based parametric model conflicts both with the required determinism and the requirement for fast re-evaluation. Therefore different structures shall be designed for the different kinds of requirements. Table 4.4 summarizes the minimal requirements for the differents kinds of parametric whole/part modelling.

whole/part modelling	occurrence specification (positioning)	occurrence specification (dimensioning)	occurrence accessibility
insertion of standard part in a product	mapping of the reference coordinate system of the part on some axis_2_placement parametrically defined in the embedding instance	parameter value = expression of the embedding context	coarse grain (e.g., mapped_item)
insertion of an equality-based parametric contour in a 3D model	constraint with (projected) items of the embedding instance	internal variable value = expression of the embedding context	fine grain: every item shall be accessible (e.g., for sweeping)
parametric assembly design (solid model)	constraints (generally oriented) between the parts of the assembly	few (possibly oriented constraints by means of expression)	medium grain: only the item referenced for positioning constraints shall be accessible
feature-based modelling	topological mapping	parameter value = expression of the embedding context	medium/fine grain (all the topology invariants, e.g., faces, invariant paths, ... shall be accessible)

Table 4.4 Minimal requirements for the different kinds of parametric whole/part modelling.

In our framework, to design a data model structure for one kind of whole/part modelling means the following:

•1) to define how the instance is modelled (e.g., as the **current_instance** of the class level or as a separate entity);
•2) to define how the occurrence is represented in the **current_instance** representation of the embedding instance (e.g., as a **mapped_item**, or as a **representation_relationship_with_transformation**, both from [ISO 10303-43]).

•3) to define the content of the **parametric_reference** that provide for reference to the content of the occurrence,
•4) to define the constraint that create the occurrence, and
•5) to define the other constraint that may access to the **parametric_reference**s and **variable**s generated by the occurrence creation.

We propose in the next section a complete data model for the insertion of a functionally-defined parametric occurrence, that is intended to be globally accessed within the context of an embedding parametric instance. This data model addresses the minimal requirement for the insertion of a (parametrically-defined) standard part within a parametrically-defined product. We then discuss in the next section 4.3.2 how some other requirement may be addressed in the context of our proposed framework.

4.3.1 Parametric functional mapping

4.3.1.1Representation of the instance level

In functional parametric modelling the **representation** of an instance is completely characterised by the value of its **parameter**s. Therefore the data structure of an instance representation, called a **pm_representation_instance**, *may* be completely different from the parametric model it corresponds to. Provide that the instance refers to its source parametric model and contains the value for its parameters, the constraints need not to be duplicated within the instance: the same representation, with its complete constraint structure and logical representation, may be generated again just by assigning the instance parameter values to the parametric model parameters. This allows to have different instances referring to the same parametric model. Such instances may have completely different parameter values from the **current_instance** represented in the **parametric_model**. It shall be underlined that this approach is the usual approach in the class/instance object oriented paradigm. The instance only contains its attribute values. The behaviour is factored at the class level.

Moreover, if some permanent names are assigned both to the parametric model and to its parameters (using e.g., [ISO DIS 13584-42]), the parametric model itself may not be represented in the same exchange context.

Finally, if we assume that the parametric model is available on the receiving site (or exchanged in the same exchange context as its instances and possibly occurrences), the content of the instance **representation**s do not need to be represented: they may be generated again on the receiving site. To ensure that such an "empty" representation is nevertheless a **representation** conforming to [ISO 10303-43], and that the instance may be mapped (as an occurrence) within the context of its embedding instance, we propose that it contains one item in its **item**s attribute: the **axis_2_placement** that represents the reference coordinate system of the instance.

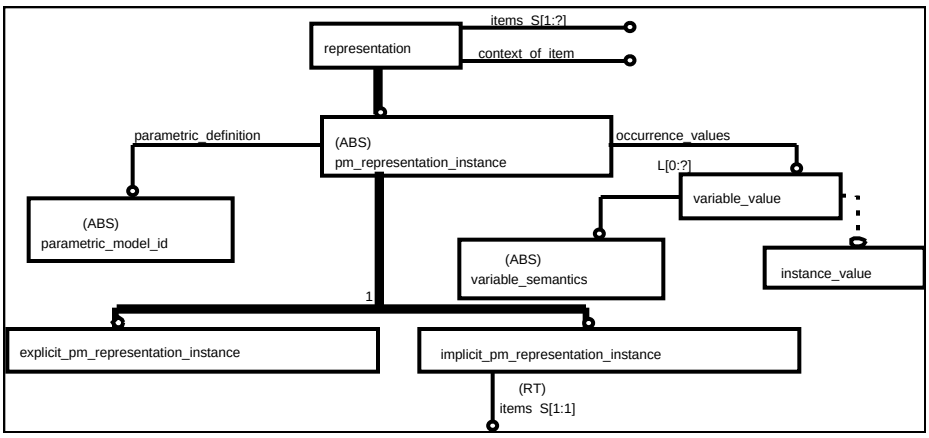


Fig. 4.5 Proposed information model for a functional parametric model occurrence

Figure 4.5 presents the proposed information model for an instance of a (functional) parametric model that is intended to be separated from its source parametric model. The abstract **parametric_model_id** entity enables to reference either a **parametric_model** or an externally defined parametric model. The **explicit_pm_representation_instance** contains, in its **items** inherited attributes, all the **representation_item**s that represent the instance. The **implicit_pm_representation_instance** subtype only contains, in its **items** inherited attribute, an **axis_2_placement**.

4.3.1.2Mapping of the occurrence

At the **current_instance** level of the embedding **parametric_model**, the occurrence insertion is represented as a **mapped_item** [ISO 10303-43] of which the **mapping_source** contains the **pm_representation_instance** as its **mapped_representation**, and the occurrence **axis_2_placement** reference coordinate system as its **mapping_origin**.

4.3.1.3Parametric reference

The occurrence may only be accessed globally (for, e.g., inserting a replica) from the embedding instance.

```
ENTITY mapped_occurrence_ref
SUBTYPE OF (geometry_parametric_reference);
   SELF\parametric_reference.physical_items: LIST[1:1]OF mapped_item;
   occurrence_parameters: LIST[0:?]OF variables;
   position: OPTIONAL axis_2_placement_ref;
WHERE
   (*not exists recorded_logical_item*)
END_ENTITY
```

Figure 4.6 Parametric_reference for a functional parametric model occurrence

This **parametric_reference** provides access (through its **referencable_items** derived attribute) to the variables that represent the occurrence parameters (they may either be used in latter constraints or, if the embedding model is not a functional one, defined through constraints), and, possibly, if it is created during the occurrence insertion (e.g. because the user has defined interactively the positioning of the part by dragging it) to the **axis_2_placement** on which the instance **mapping_origin** shall be mapped. Obviously, the occurrence parameters shall be the same as the one of the corresponding instance. Changing these parameters would generate a new instance, and then its mapping as the new representation of the occurrence. Finally, a where rule specifies that the logical item shall be the same as the physical item: the occurrence cannot be changed but by its parameter values.

4.3.1.4Parametric definition

A **functional_occurrence_definition** is a **constraint** that specifies the occurrence to be embedded within the context of some instance. It specifies, by a value of i**parametric_model_id**, the referenced parametric model. Its may define the position of the occurrence as an existing **axis_2_placement** computed by a **parametric_ref_expression**, (else an **axis_2_placement_ref** is generated by the created **mapped_occurrence_ref** for latter positioning through constraints). It may also specify, as a list of optional **expression**s, some of the occurrence parameter values. In the embedding parametric model, the occurrence parameters may then be further specified through others constraints.

A **functional_occurrence_definition** also contains a subtype, called **functional_occurrence_functional_definition**, that is a **parametric_function**. This subtype shall be used

if the embedding parametric model is a functional one. This subtype completely specifies the mapped occurrence. It specifies all the values of the occurrence parameters by means of a list of **expressions**, and the position of the occurrence as a **parametric_ref_expression**. If the occurrence is interactively positioned, the system generates as a separate parametric function, the canonical definition of the **axis_2_placement**.

```
ENTITY funtional_occurrence_definition
   SUPERTYPE OF (funtional_occurrence_functional_definition)
   SUBTYPE OF (constraint);
   SELF\constraint.defined: LIST[1:1]OF mapped_occurrence_ref;
     parametric_model_source: parametric_model_id;
     occurrence_dimensionning: LIST[0:?] OF OPTIONAL expressions;
     position: OPTIONAL parametric_ref_expression;
   DERIVE
   SELF\constraint.assumed: ...
   WHERE...
END_ENTITY;

ENTITY funtional_occurrence_functional_definiton
   SUBTYPE OF (funtional_occurrence_definiton, parametric_function);
   SELF\funtional_occurrence_definition.occurrence_dimensionning
                        : LIST[0:?] OF expressions;
   SELF\funtional_occurrence_definition.position
                        : parametric_ref_expression;
END_ENTITY;
```

Fig. 4.7 The parametric functions that inserts an occurrence of a functional parametric model in an embedding parametric model

4.3.1.5User dialogue

The user dialogue issues are outside the scope of this data model. Nevertheless, checking whether, or not, some user dialogue may be defined to populate this data model enables to validate it from an activity model point of view. We briefly outline the different steps of an example of creation dialogue:

1 - The user selects the **functional_occurrence_definition** command.

2 - The **parametric_model** is identified by some means (menu, name,...).

3 - An occurrence is instanciated, with the dimensions of the **current_instance** in the referenced parametric model. The system generates the **pm_representation_instance** and its **representation_map**. It maps it on (e.g.,) the reference coordinate system of the embedding instance and built the corresponding **mapped_item**. The **mapped_occurrence_ref** is created and initialised. Then the occurrence is displayed.

4 - The user defines the location of the occurrence either by picking up some (referencable) **axis_2_placement** or by building a **parametric_ref_expression**, or by dragging the existing occurrence. The **mapped_occurrence_ref** and **mapped_item** entity are up-dated accordingly.
If the selected command was **functional_occurrence_definition**, the process resulting from this command is achieved. The user may, latter on, assign constraints to the occurrence parameters (ensuring its latter re-evaluation).
If the selected command was **functional_occurrence_functional_definition**, the following step is performed.

5 - The user select successively, and, by some means (menu, name, picking a dimension line), each parameter, then, using the display calculator, its value is specified as a **generic_expression**.

4.3.2 Support of the other whole/part requirements

We briefly discuss, in this section, how the other requirements for whole/part parametric modelling may be addressed at the three levels of the representation of the instance to be mapped, of the occurrence mapping, and at the level of the dynamic context (i.e., the set of **parametric_reference** and **variable**s that enable access to the occurrence).

4.3.2.1Representation of an instance

When a parametric model is not functional (what is the general case for representation of a product involved in an assembly, or for an equality-based parametric contour) or when an access shall be provided to the internal structure of the occurrence while ensuring the correctness of the parametric whole/part assembly re-evaluation, the instance shall not only contain its representation, but also the **parametric_reference**s required to encapsulate the invariant of its representation.

These **parametric_reference**s shall be:
- inserted in the context of the embedding instance to provide for reference, and
- associated one to one with the **parametric_reference**s of the parametric model that defined the occurrence to provide for occurrence re-evaluation.

Two approaches may be considered to represent such an association:
•1 - to give a permanent name, (e.g., an index number), to each **parametric_reference** of a parametric model, and to use this permanent name in each occurrence (and embedding instance) to ensure the matching.
•2 - to directly refer (i.e., through their entity names) to the **parametric_reference** of the source parametric model.

The first approach might allow to separate the parametric model from its occurrence. It might be usefull (and might be introduced in our framework) for functional parametric models that describe standard parts in a library. For the other kinds of parametric whole/part modelling, this separation does not seem to be required. The parametric model is usually considered as available in the same exchange context as its occurrences, and it may be directly referenced. Moreover, both for assembly design and for equality-based contour, only one dimensioning of every occurrence generally exist: precisely the size of the **current_instance** of the parametric model itself.

Therefore, for these two applications (and unlike in section 4.3.1), the **parametric_model** itself may be considered *as the instance representation.* For these two cases of application, to separate the instance from the parametric model itself seems useless. Taking into account that (through its current instance) the **parametric_model** *is a* **representation**, we propose to consider that the **parametric_model** itself is the instance **representation** (and therefore to duplicate the parametric model if different instances are needed).

4.3.2.2Representation of the occurrence

When inserted in an embedding instance, this **representation** is either mapped (for assembly design), or transformed, representation item by representation item (for equality-based contour) in the embedding representation. In the latter case, the whole set of **parametric_reference**s of the source parametric model are duplicated (with a reference to their source) in the dynamic context of the embedding instance.

Using the **mapped_item** construct for assembly design avoid to duplicate the whole representation of the constituant component, but it does not allow to define to positioning of the component through constraints. The solution we suggest is the following. When the user want to specify a constraint between two **representation_item**s of two different components, the system identifies (in the corresponding parametric model) the **parametric_reference**s they correspond to. It instanciates the corresponding **parametric_reference** within the dynamic context of the assembly, and its **representation_item** in the **current_instance** of the assembly. This **representation_item** is defined in the same **representation_context** as the component, therefore the constraints enable a constraint-based positioning

of the product.

Regarding feature-based modelling, for both linear and revolving sweeps (but not for modifying features such that rounding, fleeting [Hoffman 92]) the global logical form of the feature (whether it is a protrusion or a cut) is recorded in the logical **representation** of the embedding instance. The physical representation, that correspond to the **current_instance** only contains the low-level B-rep entities that represent the result of the feature.

4.3.2.3The dynamic context level

For equality-based contour, the dynamic context level contains a **parametric_reference** entity for every **representation_item**s of the contour, and for every points of the face the contour belongs to.

For assembly design, the dynamic context may contain only **parametric_reference**s for those entities which are referenced by user-defined positioning **constraint**s (see section 4.3.2.2).

It is for feature-based modelling that the dynamic context proves the more useful:
•1) it allow a fine grain access to the *logical* entity that constitute the feature, and
•2) it records the feature collision [Summer 91] [Hoffman 92] in which the feature is involved.

In a B-rep modeller, a **feature_ref** defines an ordered list of **face_ref**s, a set of **feature_collision_ref**s and a set of **datum_ref**s. It refers through its **recorded_logical_items** attribute to a **shell** stored in the logical representation and do not refer to any physical representation (on the receiving system, the physical representation may be represented as e. g., a selective geometric complex [Rossignac 89] if the system supports such a representation). A **face_ref**s defines **loop_ref**s and refers both to a logical **face** and to a set of physical **face**s. The same structure is defined for the lower level topological items. This structure enables to specify constraints (or parametric functions) on the logical faces, and not on the low-level set of faces that are represented in the physical model.

4.4Constraint-based definition

The third kinds of parametric functions required for parametric modelling are the functions that create one (or several) new **representation_item**s through a number of constraints with other pre-existing **representation_item**s. We call these parametric functions **constraint_based_parametric_definition**.

To design a standard for **constraint_based_parametric_definition** needs to address two problems. (1) How to select the standard **constraint_based_parametric_definition**s. (2) How to remove ambiguity for the ambiguous geometric construct.

Concerning the first point, no definitive solution may be frozen. New functions will continuously be developed, and new exchange requirements will emerge. Our proposed framework enables to integrate such a progressive standardisation process. Moreover, to allow the exchange of data models that contain some system-specific **parametric_function**s, two mechanisms may be used. The first one, when it is feasible, consists in expressing the system-specific parametric construct in terms of the existing **parametric_function**s and **generic_expression**s. The second one is to use the following predefined parametric constraint and function:

```
ENTITY non_standard_constraint
SUBTYPE OF (parametric_function);
   utilities: LIST [1:?] OF generic_expressions;
   supplier_name: label;
   software_product: label;
   description: text
DERIVE
```

```
   used_var:SET  OF  generic_variable:=used_gen_variables_in_aggregate(SELF.utili-
ties);
   used_ref:SET  OF  parametric_reference:=used_param_ref_in_aggregate(SELF.utili-
ties);
   SELF\constraint.assumed:LIST[0:?] OF param_ref_or_var
                         :=set_to_list(SELF.used_ref + SELF.used_var);
END_ENTITY;

ENTITY non_standard_parametric_function
   SUBTYPE OF (parametric_function, non_standard_constraint);
END_ENTITY;
```

These entities enable both to store and to exchange proprietary parametric model in the standard exchange format, while insuring the formal correctness of the data model (e.g., composition order of the parametric function). The user dialogue to be implemented on the receiving system to interpret unknown extensions is outside the scope of this data model.

As far as the Parts Library requirements are concerned, the minimal set of needed **constraint_based_parametric_definition**s has already been identified. This set only addresses 2D et CSG solid parametric representation. We suggest to use this set of constraint-based parametric functions, together with the **canonical_parametric_function** defined in section 4.2, as a first resource schema for parametric geometry, just extending it with parametric functions for lines and circles (in ISO DIS 13584, all the 2D curve-oriented parametric functions specify bounded-curves). Figure 4.8 present the list of the ISO DIS 13584-31 functions.

Direction	
Dir_2_Pnt	Direction vector defined by two points
Dir_2_Dir_Angle	Direction vector defined by two directions and an angle
Dir_A2p_X	X direction from an axis2_placement
Dir_A2p_Y	Y direction from an axis2_placement
Dir_A2p_Z	Z direction from an axis2_placement
Axis2_placement (Local Coordinate System)	
A2p_3_Pnt	Axis2_placement by 3 point
A2p_2_Dir	Axis2_placement by 2 direction and 1 point
A2p_2_Dir_Xy	Axis2_placement by 2 direction (Ox) and (Oy) and 1 point
A2p_Position_Relative	Axis2_placement positioning relative
A2p_Ref_Sys	Axis2_placement by reference system
Points with numeric definition	
Pnt_Cartesian_Relative	Point cartesian relative
Pnt_Polar_Absolute	Point polar absolute
Pnt_Polar_Relative	Point polar relative
Pnt_Cylinder_Absolute	Point cylinder absolute
Pnt_Cylinder_Relative	Point cylinder relative
Points with constrained based definition	
Pnt_Begin_Ent	Point at begin of a curve entity
Pnt_End_Ent	Point at end of a curve entity
Pnt_Intersection_2_Ent	Point at intersection of two entities
Pnt_Tangential_Arc	Point tangential to a circular arc with one direction
Pnt_Centre_Arc	Point at centre of a circular arc

Pnt_Middle_Ent	Point in the middle of a basic entity
Pnt_Projection_Ent	Point as a projection on an entity
Pnt_Projection_A2p	Point as a projection on an a2p entity
Trimmed line	
Lin_2_Pnt	Line segment between two points
Lin_Pnt_Length_Dir	Line segment by start point, length and direction
Lin_Tangential_Arc	Line segment tangential to circular arc from a point
Lin_Tangential_2_Arc	Line segment tangential to two Circular arc
Lin_Chamfer_2_Lin	Line segment as chamfer of two line segments (modifying feature)
Circular_arc (trimmed_circle)	
Arc_3_Pnt	Circular arc by three points
Arc_Rad_2_Angle_A2p	Circular arc by radius and two angles
Arc_Rad_3_Pnt	Circular arc by radius and three points
Arc_Rad_2_Pnt_A2p	Circular arc by radius, two points and axis2_placement
Arc_Fillet_2_Ent	Circular arc as fillet between two entities (modifying feature)
Arc_Tangential_2_Ent	Circular arc tangential to two entities
Arc_Rad_2_Ent	Circular arc defined by its radius and two entities
Arc_3_Ent	Circular arc defined by three entities
Conics and trimmed conics	
Ellipse_2_Diameter_A2p	Ellipse by two diameters and placement
Elc_Gen	Elliptical arc generation
Hyp_Gen	Hyperbolical arc generation
Par_Gen	Parabolical arc generation
Polyline	
Pln_Cartesian_Coordinate	Polyline by list ofcartesian coordinates
Bounded_surface_curve (planar)	
Ctr_Gen	Generation of a contour from curves
Curve_bounded_surface (planar)	
Aps_Gen	Generation of a planar surface from its boundaries
Special purpose CSG solid (pipe entity)	
Sld_Pipe	Generation of a pipe
Group structure (geometric_set)	
Create_Grp	Create group
Close_Grp	Close group
Reopen_Grp	Reopen group
Remove_Ent_Grp	Remove entity from group
Gather_Ent_Grp	Gathering entity into new group
Add_Ent_Grp	Adding entity into group
Geometric_set_replica	
Dup_Mirror_Ent	Duplicate and mirror entity or set
Dup_Shift_Dir_Ent	Duplicate and shift an entity or set defined by a direction and length
Dup_Shift_Displacement_Ent	Duplicate and shift an entity or set defined by displacements
Dup_Rotate_Ent	Duplicate and rotate an entity or set

Fig. 4.8 Constraint-based parametric functions defined by [ISO DIS 13584-31]

Concerning ambiguity removal, [ISO DIS 13584-31] specifies very precisely the ambiguity removal process on the basis of entity orientation. Therefore, it would be possible to refer to this specification for the description of the standard **constraint_based_parametric_definition**s. The only changes would be:

•1) to add for each reference (in the **assumed** list) to an entity, an additional attribute **same_sense** *: Boolean* (in interactive design, this attribute will be computed by the interface, cf. figure 2.3), and

•2) in the specification of the parametric function, to remove all the duplication of entities used as input parameter (the goal of ISO DIS 13584 is to drop any parametric link between entities, when the goal of the parametric data model is precisely to record then).

4.5 Alternative and repetitive shape aspects

The fourth kinds of parametric functions required for parametric modelling is the capability to capture, in the same parametric model, various possible instances that contain alternative or repetitive shape aspects. This capability is intended to provide functional parametric data model with the same expressive power as variant programming [ROLLER 91], largely used in the previous generation of CAD system.

In the present parametric CAD systems, such a capability is generally restricted to the concept of "patterns" (see figure 4.9), where the same feature is repeated according to some predefined pattern structure. Few systems for instance are able to design a block with a set of holes, the number of holes depending on the length of the block. None of them, to our knowledge, are able to design general purpose recurrence-based repetitive patterns such that each shape aspect depends on the previous one (see figure 4.9). An approach based on context structure has already been proposed [GIRARD 92] [PIERRA 94b]. It might be modelled within the framework proposed in this paper.

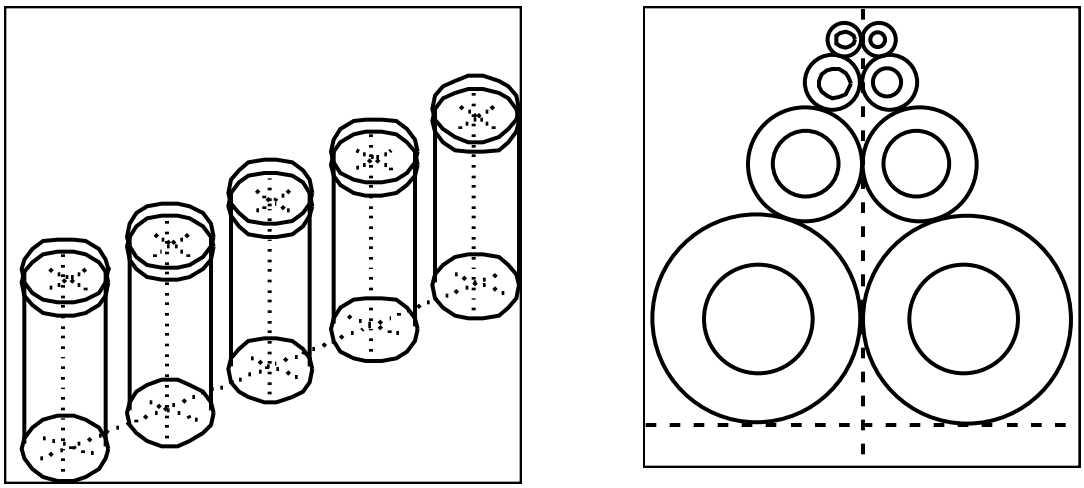


Fig. 4.9 Simple patterns and recurrence-based repetitive shape aspects

5 Implementation and further research

With the support of the ESPRIT PLUS project, we have developed a 2D functional parametric system based on the architecture proposed in this paper. The goal was both to extend the expressive power of the existing parametric systems towards alternative and repetitive shape aspect specification and to investigate the rela-

tionships between design history (the systematic and implicit capture of all the design process) and functional parametric model as defined in section 2. Through a structure of the dynamic context, this system support parametric whole/part modelling and any kind of alternative or repetitive shape, even those where a general recurrence relationship exist within the shape repetitive pattern. In this system, the parametric definition is captured during the design, but we have developed a powerful graphical modification ("debugging") environment.

At any phase of the design process, it is feasible to modify any constraint or entity. When the user pick up an entity for modification, the system returns to the state where this entity was created. It is then possible to change (graphically) its nature or its constructive process, to insert at this place an additional set of constructive steps, or to remove a set of entities (and their functional specification). When any modification is performed, the system checks the remaining part of the parametric specification for consistency and, when some parametric references belonging to the **assumed** part of some latter constraint have been removed, asks for a new parametric specification of the corresponding entities. All these feature demonstrate the existing difference between functional parametrics and design history.

An interesting feature of our approach is that the implementation of parametric capabilities on some existing CAD system requires very few modification of the kernel CAD system [Girard 92] [Potier 95]. We are now developing some prototype implementation on a B-rep modeller, to validate our proposal for feature-based modelling, and a post processor that generates from our parametric internal data model an exchange file conforming to the EXPRESS specification proposed in this paper.

Regarding the development of a complete pre-normative proposal for the exchange of parametric data model, we are now discussing in the SC4 standardization arena the proposed framework. Once a framework is agreed, the development of such a document requires a very precise specification of all the parametric functions and constraints allowed for exchange. We plane to develop this specification for the (restricted) part that address the requirements of ISO 13584: functional parametric model for 2D and solid geometry supporting general alternative et repetitive shapes. We hope to collaborate with other projects for the specification of equality-based parametric capabilities and the refinement of feature-based modelling.

Conclusion

Over the last few years, a lot of progress have been achieved in product data technology. The first release of STEP has been issued, and more and more companies invest in its practical use. The P-LIB standard that provides for electronic data dictionary modelling and for parts library management and exchange is emerging. Several prototype implementations have already been developed. A masterpiece is still missing: the capability to model and to exchange parametric models, either to capture, together with a product, its design intend, or, to store, in a parts library, a unique parametric model able to generate the shapes, or other representations, of all the parts of a part family. The requirements appear as slightly different. The P-LIB requirements are restrictive but well defined. The STEP requirements are much broader but still not precise.

In this paper we have presented a global framework intended to fulfil both kinds of requirements. Its general architecture should be able to support the foreseeable progress of the parametric technology, and a progressive standardisation process. Its already achieved information models fulfil the requirements defined for parts library modelling and exchange.

This framework is based on three ideas. The first idea is that a parametric product/part *is a* product. To associate to a product its parametric definition (or definitions) does not change the product, nor its data model. It only enlarges its information model. By defining a parametric (representation) model as a subtype of representation, and by orienting the references from the parametric definition to the product data, i.e., without any change of the product data itself, we have shown in this paper that parametric capabilities might be introduced in the context of STEP and P-LIB without any change of the information models (e.g., Integrated

Resources, Application Protocols) already developed.

The second idea is that, even if parametric encapsulates in fact a process, this process is usually stored in the database of parametric systems. Therefore, it should be feasible to capture this implicit process in a data specification language. Using a meta-programming approach, where the process is captured through an abstract syntax tree, we have proved that parametric model may be captured and exchanged without any extension neither of the EXPRESS language, nor to its implementation forms (Physical files, SDAI).

The third idea is to try to remove the foreseeable limitations of the parametric technology through an analogy with similar domains. In programming languages, the major limitation with e.g., FORTRAN was the static context management. The introduction of a fine (and dynamic) context management provided the modern capabilities for, e.g., recursivity, dynamic variables and object oriented environment. In this paper we have proposed to introduce a concept of dynamic context as an abstraction level between the parametric specification and the low level physical geometric model. This structure provides for feature-based modelling and should support the foreseeable progress towards the specification of alternative shapes, general repetitive shapes, and, possibly, recursive shapes.

This framework, which improves a first proposal presented in [Pierra 94a] [Pierra 94b], is intended to be circulated in the standardisation arena before summer 1996. Improved by this international review process, it is hoped it may emerge as a standard in the very next years.

8. References.

[Ait-Ameur 95]Y. Ait-Ameur, F. Besnard, P. Girard, G. Pierra, J.C. Potier "Specification and Metaprogramming in the EXPRESS Language", *In Intern. Conference on Software Engineering and Knowledge Engineering SEKE'95 (IEEE - ACM Sigsoft), Rockville, USA,* June 1995 *pp. 181-189.*

[Aldefeld 88]B. Aldefeld, "Variation of geometries based on a geometric-reasoning method". Computer Aided Design, 20, 3, 1988, pp. 65-72.

[Bouma 95] W. Bouma, I. Fudos, C. Hoffmann, J. Cai, R. Paige. "Geometric Constraint Solver". Computer Aided Design, Vol 27, Number 6 june 1995, pp 487-501

[Chou 84] S.-C. Chou "Proving-Elementary Geometry Theorems Using Wu's Algorithm", Contemporary Mathematics, 29, AMS, 1984, 243-286.

[Chou 87] S.-C. Chou, W.F. Schelter & J.-G. Yang "Characteristic Sets and Gröbner Bases in Geometry Theorem Proving", Workshop on. Computer-Aided Geometric Reasoning, INRIA, Sophia-Antipolis, 1987, pp.29-56.

[Dufour 90]J.F. Dufour "Programmation et résolution de problèmes de construction géométrique", BIGRE ,67, Jan. 1990, pp..136-147.

[Girard 90] P. Girard, G. Pierra, L. Guittet, "End User Programming Environments: Interactive Programming-On-Example in CAD Parametric Design", EUROGRAPHICS'90, Ed. North-Holland, Montreux (Septembre 1990), p. 261-274.

[Girard 92]P. Girard "Environnement de programmation pour non programmeurs et Paramétrage en Conception Assistée par Ordinateur : Le Système Like", Ph D thesis, University of Poitiers, November 1992.

[Girard 93]Girard, P., Pierra, G. "Command Recording versus Parametric and Variational Systems, and old/new third way of parametrizing CAD models by End Users". COMPEURO'93 (IEEE-SEE), 1993, pp. 194-200.

[Hillyard 78]R. Hillyard, T. Braid "Analysis of dimensions and tolerances in computer-aided mechanical design". Computer Aided Design, 10, 3, 1978, pp. 161-166.

[Halbert 84]D. Halbert "Programming by example. PhD". Thesis, Berkeley Univ., California, 1984, pp.121.

[Hoffman 92]C. Hoffman, R. Juan "Erep, an editable, high-level representation for geometric design and analysis" in Geometric Modeling for Product Realization. P. Wilson, M. Wozny, M. Pratt, eds. North Holland, 1992, pp 129-164

[ISO 10303-11] ISO 10303-11, Industrial Automation Systems and Integration, Product Data Representation and Exchange, "The EXPRESS language reference manual", ISO, Geneva, 1994

[ISO 10303-21] ISO 10303-21,Industrial Automation Systems and Integration, Product Data Representation and Exchange, "Clear text encoding of the exchange structure", ISO, Geneva, 1994

[ISO 10303-41] ISO 10303-41, Industrial Automation Systems and Integration, Product Data Representation and Exchange, "Fundamentals of product description and support", ISO, Geneva, 1994

[ISO 10303-42] ISO 10303-42, Industrial Automation Systems and Integration, Product Data Representation and Exchange, "Integrated generic resources: geometric and topological representation", ISO, Geneva, 1994

[ISO 10303-43] ISO 10303-43, Industrial Automation Systems and Integration, Product Data Representation and Exchange, "Representation structure", ISO, Geneva, 1994

[ISO DIS 13584-20] ISO DIS-13584-20: Industrial Automation Systems and Integration, Parts Library, "General Resources", ISO, Geneva, 1996

[ISO CD 13584-24] ISO CD-13584-24: Industrial Automation Systems and Integration, Parts Library, "Logical Model of supplier library", G. Pierra, Y. Ait-Ameur, Eds, (ISO TC184/SC4/WG2 N209)

[ISO DIS 13584-31] ISO DIS-13584-31: Industrial Automation Systems and Integration, Parts Library, "Geometric Programming interface", ISO, Geneve, 1996

[ISO DIS 13584-42] ISO DIS-13584-42: Industrial Automation Systems and Integration, Parts Library, "Methodology for structuring part families", ISO, Geneve, 1996

[Kin 89] N. Kin "PictureEditor: A 2D Picture Editing System Based on Geometric Constructions and Constraints". Proc. Comp. Graphics Int.'89, Leeds, Springer-Verlag, 1989, pp.193-208.

[Kramer 92]G. Kramer. "Solving Geometric Constraint Systems" MIT press, 1992

[Lee 82] K. Lee, G. Andrews "Inference of the positions of components in an assembly : Part 2". Computer Aided Design, 17, 1, 1985, pp. 20-24.

[Light 82] R. Light, D. Gossard "Modification of geometric models through variational geometry". Computer Aided Design, 14, 4, 1982, pp. 209-214.

[Myers 86]B. Myers "Visual Programming, Programming by Examples, and Program Visualization : A Taxonomy". Proc. of SIGCHI 86, Human Factors in Computer Systems, New-York, 1986, pp. 59-66.

[Myers 90]B. Myers "Taxonomies of Visual Programming and Program Visualization". J. of Visual Lang. and Comp., 1, 1990, 97-123.

[Owen 91] J. Owen. "Algebraic solution for geometry from dimensional constraints". In ACM Symp. FOund. of Solid Modeling, pp 397-407 , Austin, Tex, 1991

[Pierra 94a] G. Pierra "Parametric product modelling for STEP and Parts Library (V0.3)", ISO-STEP meeting, Davos, Mai 1994, ISO/TC 184/SC4/WG2 N183, July 1994, 107 p.

[Pierra 94b] G. Pierra, J.C. Potier, P. Girard "The EBP system : Example Based Programming for parametric design", Workshop on Graphic and Modeling In Science ant Technology, Coimbra, 27-28 june 1994, in: Springer Verlag Series, 1996.

[Pierra 94c]G. Pierra "Modelling classes of pre-existing components in a CIM perspective: the ISO 13584/ ENV 400014 Approach", Proc. of PDT Days'94, in: Revue internationale de CFAO et d'Infographie, vol. 9, n 3, 1994 , pp. 435-454.

[Potier 95] J.C. Potier "Contribution à la notion de programmation par démonstration. Conception sur exemple, mise au point et génération de programmes portables de géométrie paramétrée dans le système EBP" PhD thesis, University of Poitiers, France, 1995

[Roller 89]D. Roller, F. Schonek, A. Verroust "Dimension-driven geometry in CAD : a survey" in : Theory and Practice on Geometric Modeling", Springer Verlag, 1989, pp. 509-523.

[Roller 91] D. Roller "Advanced methods for parametric design" in Hagen, H and Roller, D (Eds) Geometric Modelling, Methods and Applications, Springer-Verlag (1991) pp 251-266

[Roller 95] D. Roller "Solid Modelling with Constraint Form Features" in Computing, Springer-Verlag 95, pp 275-284

[Rossignac 89] J. R. Rossignac, M. A. O'Connor "SCG: a dimension-independent model for pointsets with internal structures and incomplete boundaries" in: Geometric Modelling for Product Engineering, Elsevier Science Publishers, 1990, pp 149-172

[Solano 94]L. Solano, P. Brunet "Constructive Constraint-based model for parametric CAD systems" in Computer-Aided Design volume 26 Number 8 pp 614-621 1994

[Summer 91] R.M. Summer "Feature and face editing in a hybrid solid modeler" in ACM/SIGGRAPH Symposium on Solid Modeling Foundations and CAD/CAM Applications, June 1991.

[Sunde 87]G. Sunde "A CAD system with declarative Specification of Shape". Proc. of EuroGraphigs Workshop on Intelligent CAD Systems, Noodwijkerhout, The Nederlands, 21-24 April, 1987.

[Sutherland 63] I. E. Sutherland "A Man-Machine Graphical Communication System", Proc. of AFIPS Spring Joint Comp. Conf., 23, 1963, pp. 329-346.

[Verroust 90]A. Verroust "Construction d'objets géométriques définis par des contraintes". BIGRE, 67, Jan. 1990, pp. 62-74